MacTech Magazine
Vol. 16, No. 8 • August 2000

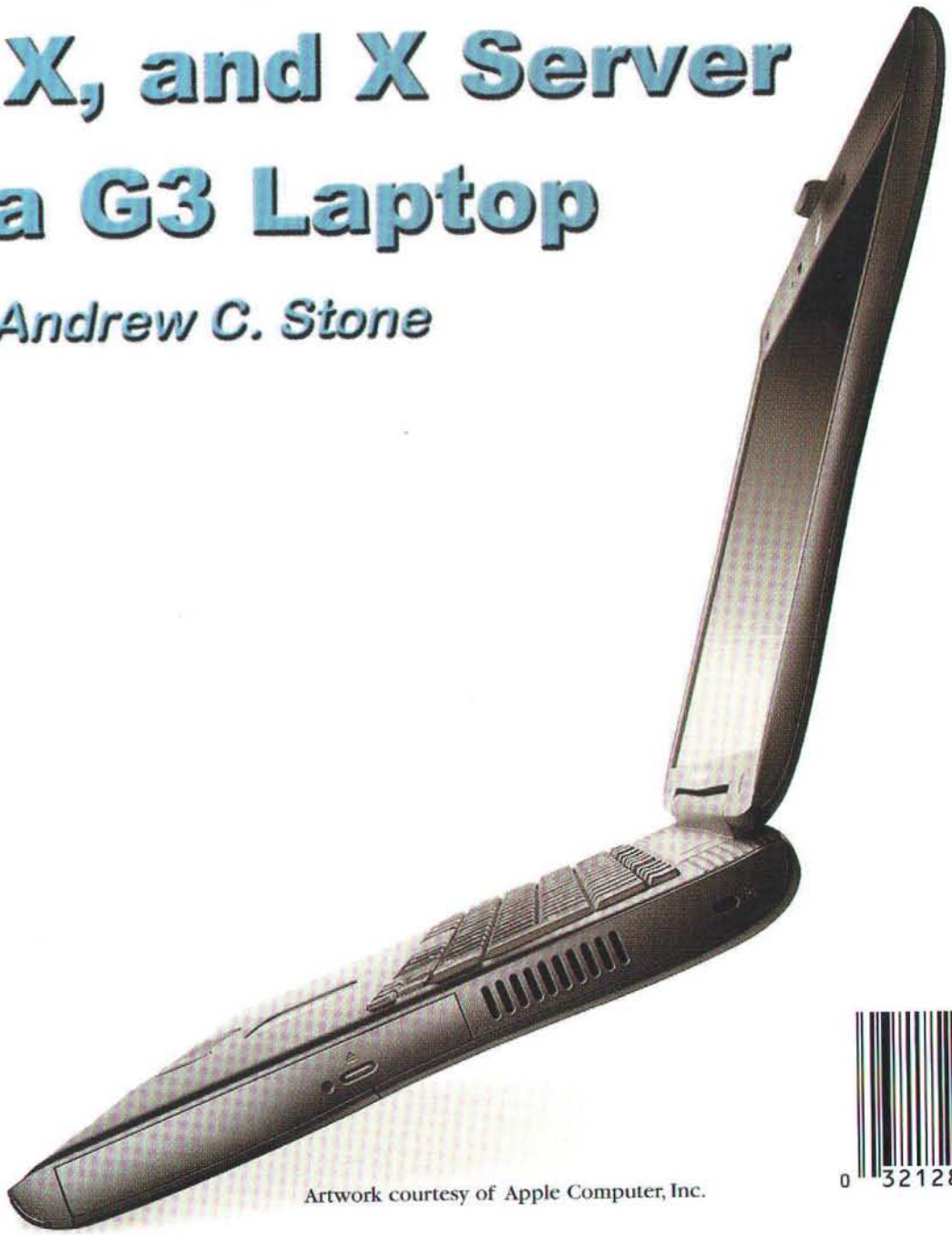
MacTech®

The Journal of Macintosh Technology and Development

Review: Lasso Studio for Dreamweaver

Installing Mac OS 9, OS X, and X Server on a G3 Laptop

By Andrew C. Stone



Artwork courtesy of Apple Computer, Inc.

$8.95 US
$12.95 Canada
ISSN 1067-8360
Printed in U.S.A.

08
0 32128 74887 8

Keeping Mac dreams alive since 1993.

Two great operating systems, one CodeWarrior.

Mac developers depended on CodeWarrior to make the platform architecture shift from 68K to PowerPC processors. Now CodeWarrior does it again, speeding your transition to the next new operating system. CodeWarrior for Mac OS, Version 6.0 supports development for both OS X and Classic Mac operating systems from a single, powerful, award-winning Integrated Development Environment. Discover how CodeWarrior for Mac OS, Version 6.0 can help you realize your Mac development dreams. Visit **www.metrowerks.com/go/mac**.

CodeWarrior®

metrowerks
Software Starts Here

"Without a doubt, the Premiere Resource Editor for the Mac OS ... A wealth of time-saving tools."
– MacUser Magazine Eddy Awards

"A distinct improvement over Apple's ResEdit."
– MacTech Magazine

"Every Mac OS developer should own a copy of Resorcerer."
– Leonard Rosenthol, Aladdin Systems

"Without Resorcerer, our localization efforts would look like a Tower of Babel. Don't do product without it!"
– Greg Galanos, CEO and President, Metrowerks

"Resorcerer's data template system is amazing."
– Bill Goodman, author of Smaller Installer and Compact Pro

"Resorcerer Rocks! Buy it, you will NOT regret it."
– Joe Zobkiw, author of ***A Fragment of Your Imagination***

"Resorcerer will pay for itself many times over in saved time and effort."
– MacUser review

"The template that disassembles 'PICT's is awesome!"
– Bill Steinberg, author of Pyro! and PBTools

"Resorcerer proved indispensible in its own creation!"
– Doug McKenna, author of Resorcerer

Resorcerer® 2

Version 2.0

The Resource Editor for the Mac™ OS Wizard

ORDERING INFO

Requires System 7.0 or greater,
1.5MB RAM, CD-ROM

Standard price: $256 (decimal)
Website price: $128 - $256
(Educational, quantity, or
other discounts available)

Includes: Electronic documentation
60-day Money-Back Guarantee
Domestic standard shipping

Payment: Check, PO's, or Visa/MC
Taxes: Colorado customers only

Extras (call, fax, or email us):
COD, FedEx, UPS Blue/Red,
International Shipping

MATHEMÆSTHETICS, INC.
PO Box 298
Boulder, CO 80306-0298 USA
Phone: (303) 440-0707
Fax: (303) 440-0504
resorcerer@mathemaesthetics.com

New in 2.0:

- **Very fast, HFS browser for viewing file tree of all volumes**
- **Extensibility for new Resorcerer Apprentices (CFM plug-ins)**
- **New AppleScript Dictionary ('aete') Apprentice Editor**
- **MacOS 8 Appearance Manager-savvy Control Editor**
- **PowerPlant text traits and menu command support**
- **Complete AIFF sound file disassembly template**
- **Big-, little-, and even mixed-endian template parsing**
- **Auto-backup during file saves; folder attribute editing**
- **Ships with PowerPC native, fat, and 68K versions**

- Fully supported; it's easier, faster, and more productive than ResEdit
- Safer memory-based, not disk-file-based, design and operation
- All file information and common commands in one easy-to-use window
- Compares resource files, and even **edits your data forks** as well
- Visible, accumulating, editable scrap
- Searches and opens/marks/selects resources by text content
- Makes global resource ID or type changes easily and safely
- Builds resource files from simple Rez-like scripts
- Most editors DeRez directly to the clipboard
- All graphic editors support screen-copying or partial screen-copying
- Hot-linking Value Converter for editing 32 bits in a dozen formats
- Its own 32-bit List Mgr can open and edit very large data structures
- Templates can pre- and post-process any arbitrary data structure
- Includes nearly 200 templates for common system resources
- TMPLs for Installer, MacApp, QT, Balloons, AppleEvent, GX, etc.
- Full integrated support for editing color dialogs and menus
- Try out balloons, 'ictb's, lists and popups, even create C source code
- Integrated single-window Hex/Code Editor, with patching, searching
- Editors for cursors, versions, pictures, bundles, and lots more
- Relied on by thousands of Macintosh developers around the world

To order by credit card, or to get the latest news, bug fixes, updates, and apprentices, visit our website...

www.mathemaesthetics.com

MacTech®
The Journal of Macintosh Technology & Development

A publication of XPLAIN CORPORATION

How To Communicate With Us

In this electronic age, the art of communication has become both easier and more complicated. Is it any surprise that we prefer **e-mail?**

If you have any questions, feel free to call us at 805/494-9797 or fax us at 805/494-9798.

If you would like a subscription or need customer service, feel free to contact Developer Depot Customer Service at 800-MACDEV-1

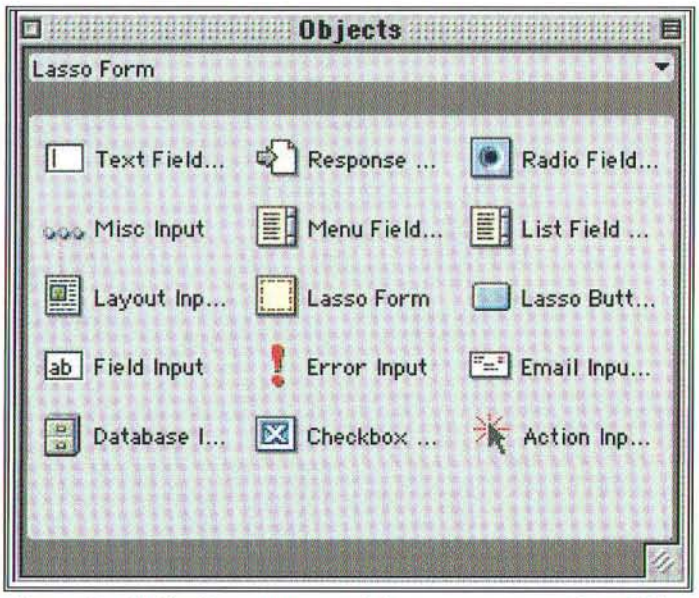
DEPARTMENTS	E-Mail/URL
Orders, Circulation, & Customer Service	cust_service@mactech.com
Press Releases	press_releases@mactech.com
Ad Sales	ad_sales@mactech.com
Editorial	editorial@mactech.com
Programmer's Challenge	prog_challenge@mactech.com
Online Support	online@mactech.com
Accounting	accounting@mactech.com
Marketing	marketing@mactech.com
General	info@mactech.com
Web Site (articles, info, URLs and more...)	http://www.mactech.com

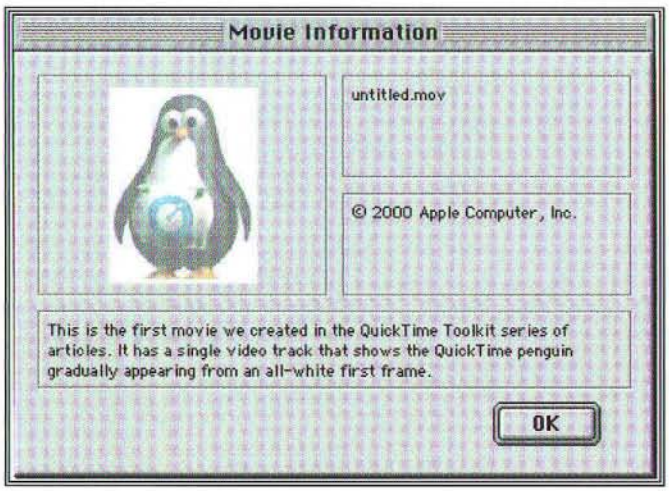
MacTech Magazine (ISSN: 1067-8360 / USPS: 010-227) is published monthly by Xplain Corporation, 850-P Hampshire Road, Westlake Village, CA 91361-2800. Voice: 805/494-9797, FAX: 805/494-9798. Domestic subscription rates are $47.00 per year. Canadian subscriptions are $59.00 per year. All other international subscriptions are $97.00 per year. Domestic source code disk subscriptions are $77 per year. All international disk subscriptions are $97.00 a year. Please remit in U.S. funds only. Periodical postage is paid at Thousand Oaks, CA and at additional mailing office. Ride-Along Enclosed.
POSTMASTER: Send address changes to **MacTech Magazine**, P.O. Box 5200, Westlake Village, CA 91359-5200.

Contents

August 2000 • Volume 16, Issue 08

Feature Articles





Columns

By Jordan Dea-Mattson

Of Tulips and Dot.COMs

A Mania for Tulips

In his book *Extraordinary Popular Delusions & the Madness of Crowds*, Charles MacKay recounts the history of the Danish Tulip Mania of the 17th Century. It is a fascinating story that has definite lessons for us today.

Tulips were originally introduced to Europe from Turkey in the mid-1500s. Their beauty made them popular among the well to do and having various varieties in your garden soon became a status symbol, if not a requirement for being considered a cultured individual. Soon, connoisseurs, desiring to one-up one another, would focus on getting the right variety of bulbs, so that their gardens would be "just right".

As you might suspect, as the popularity of tulips grew, the demand for tulip bulbs drove their prices higher and higher. Ordinary people seeing the price for individual bulbs double and then double again soon started buying them — not to have them in their gardens — but as investments.

Holland was hit particularly hard by the mania for tulips. An entire industry of tulip exchanges and brokers sprung up, and soon ordinary folks were emptying their bank accounts and mortgaging their homes to get in on this sure-fire way to riches.

A "crash" — or to use the new and improved, politically correct phrase, a "correction" — finally came, and when it did it was bloody. People lost their homes. Families were bankrupt. People's "sure thing" investments in tulip bulbs returned to being what they were originally: the source of beautiful flowers.

Today's Tulips

For the last several years — up through March of this year — anything that smelled of "dot.com", "Internet", "digital convergence", or "new economy" was the "sure thing" that would bring you untold riches. For a while — as in the Danish Tulip Mania — it seemed to hold true. You could invest in almost any "half-baked" idea — and lets be honest here, they were ideas not companies — and you would double your money in a couple of months.

But when you dug into these ideas you found that many of them weren't and wouldn't ever be ready for baking. People were launching magazines on the Internet — a very tough business, just ask our publisher — like Salon.com, and commanding valuations that you would never find placed on an established magazine with a stable subscriber base and lots of faithful advertisers. It was absurd.

Even that darling of the Internet's new economy, Amazon.com — which when you dig into it is only a very slick and efficient retailer — was and still is, even at around $30 a share, absurdly overvalued.

And like in the Danish Tulip Mania, a whole industry has grown up around the trading of our latest mania. There are web sites — many of which, in the greatest of ironies, have gone public as part of our Internet Mania — chat rooms, magazines, and mutual funds, all of which are focused on driving the Mania forward.

And, like the Danish Tulip Mania, our Internet Mania seems to be unwinding itself to an extent. As I write this column, the NASDAQ has finished a week long decline. A decline that has former Internet darlings like Amazon.com and e-toys further in the red than ever before. The mania part of the Internet seems to be — unlike the Danish Tulip Mania — imploding softly.

It Isn't All Mania

If you are reading this column, you in all probability agree with me that the Internet is going to change the way that we live across the board. You probably agree that we haven't seen but a glimmering of the beginning of the changes it will bring. This is true, but it is going to be a long-hard road to the "Promised Land". Unfortunately, it is going to be a long road with a lot of potholes along the way.

The mania isn't over. It has just subsided for a while. In the years ahead, there will be Internet related gold rushes from time to time. Gold rushes that will happen, because in large part "there is gold in them hills..." This is the fundamental difference between the Danish Tulip Mania and our present dot.com mania: if you dig hard enough, you can find value.

Making Money in a Gold Rush

In the California Gold rush of 1849 — and any of the others I have studied — for every person that struck it rich prospecting for gold, there were hundreds that lost everything. It wasn't the miners or the prospectors that made fortunes.

The folks that struck it rich in each and every gold rush were those that provided goods and services to the miners and the prospectors. It was people selling denim jeans — does the name Levi-Strauss ring a bell, food, laundry service, tools, banking, and recreation.

This is a lesson that we should keep in mind as we survey the business prospects in the "new economy". Over the last few years I have watched friends and acquaintances stream into the businesses of the "new economy" one by one. I have seen them go to one "sure thing" after another. They have gone to web-tailers, and business to business exchanges. They have joined network infrastructure companies and system integrators. By and large they are all doing well, but the ones that are most secure are the ones that are selling something to the "miners" and "prospectors" of the Internet gold rush.

If you want to make your fortune on the frontiers of the New Economy, I recommend that you focus on providing services to the prospectors with stars in their eyes and cash in their pockets. But make sure that you get cash up-front, and try and get a piece of each claim that they file. Who knows, one of them might strike a mother-load.

MT

The key to thinking different...

Works on the iMac!

The New MacHASP USB Key!

Question: Is MacHASP USB* a software security key or a sales tool?

Answer: It's both!

MacHASP USB is the world's first software protection key for the iMac. It gives you sophisticated license enforcement and comprehensive protection against illegal use ... in other words, real security.

Then it gives you a big selling advantage.

With MacHASP USB, you can license multiple software modules and applications. You can instantly unlock or upgrade them in the field. Plus you can freely distribute demos.

Bottom line: MacHASP USB locks out illegal users and unlocks your full sales potential – without getting in anyone's way. Call now to request a Developer's Kit and our newly published guide to USB features and benefits.

*For all USB-equipped Macs running an OS with USB support. Fully compatible with the ADB version of MacHASP.

Mac OS

Accelerated for Power Macintosh

USA: 1-800-223-4277
212-564-5678
Int'l: 972-3-6362222

www.aks.com/mt

ALADDIN®
KNOWLEDGE SYSTEMS LTD

Mac software protection provider, Micro Macro Technologies, becomes part of Aladdin, giving its customers even better service from the number one name.

By John C. Welch

AppleScripting Your Network

How to use Apple's second – best kept secret to make your life easier

Welcome

As you can guess, I am going to talk about the how's and why's of using AppleScript to make your life as a network administrator much easier. We'll talk about some of the reasons for using AppleScript to run networks, and some of the instances when you would, or would not want to use AppleScript. We'll also take a look at some example scripts that I use to make *my* life easier.

Background

Just to give you an idea on where I am coming from in my approach to scripting a network, I think it's important to give you an idea of what my network consists of. AER, like most scientific companies, has more than one operating system and hardware platform that it uses to get the job done. Our primary platform is Sun Solaris 7 on Sun Sparc hardware. We also make extensive use of SGI's Irix, IBM's AIX on RS/6000s, and Linux, both on Intel and PPC. Our non-Unix operating systems include Windows 98, Windows NT, Windows 2000, and OS/2 on Intel hardware, and Mac OS 8.6 and 9.0.4 on Apple hardware. My direct responsibility is as the administrator for some 70+ Mac and Wintel machines, with indirect responsibility to everything else.

So, for my needs, any scripting I do, has to be able to handle non-MacOS machines whenever possible. This makes things, well, interesting at times. But then again, that makes it fun. We also let the users, and their department heads decide for themselves what hardware to use, so any standards in that area come from the people using the equipment, instead of the people fixing it. Managing all this can be a handful, and without AppleScript, darn near impossible.

Foundations

First of all, before you can script any thing, from a network to email, you need to be familiar with what you are scripting, and its capabilities, both good and bad. That means there are some critical pieces of the picture that you must know to script your network well.

Know Your Network

This means, that for the areas you have responsibility over, you have to know things at a physical level. You should know has much about the wiring runs, and wiring closets as possible. Where possible, get a set of blueprints, and diagram where the wire runs travel, and how they get in and out of the closets. This obviously has to bow to common sense. If you have twenty buildings in seventeen countries to manage, it may be impossible to get *all* the blueprints for you network, but you should put forth a sincere effort to know where your wires are.

This also means that you need to know your network equipment. Having bright shiny G4s will save you no time if your hubs are shared 10Mbps to hundreds of users. You need to know what your network equipment is, and what it can and cannot do as well. Remember, a network is almost a living organism, and any scripting done in that environment must be done holistically. It does you no good to have a fantastic script that requires AppleTalk drive mounting on servers in another subnet, if none of your routers are passing AppleTalk.

Once you have these parts, then you need to know how your network is laid out logically as well as physically. You may be one jack over from a file server, but if your network makes

John Welch <jwelch@aer.com> is the Mac and PC Administrator for AER Inc., a weather and atmospheric science company in Cambridge, Mass. He has over fifteen years of experience at making computers work. His specialties are figuring out ways to make the Mac do what nobody thinks it can, and showing that the Mac is the superior administrative platform.

use of VLAN, (virtual LAN) switches, then your logical layout may have you many subnets away from that same server. Again, depending on what you need to automate, then the differences between logical and physical layout, and how aware of them you are make the difference between success and failure.

Know what computers and operating systems are going to be affected by your scripting. It does no good to write scripts that assume your PCs are running Windows 98 if they are actually running Windows 2000. On the Mac side, differences in support for various AppleScript features between versions of the MacOS can complicate your life to an amazing degree if they catch you by surprise. Along with this, take a little time to find out what the users thinks that they are using their computers for. The answer might surprise you, as users have an uncanny ability to customize their computers to work best for their needs. This is not bad or good, but rather something you need to be aware of. That computer is their tool first, and yours second. Make sure that by making your life easier, you don't make theirs harder.

All of the above boils down to knowing how your network is designed and installed, and why it is set up the way it is. By taking the time to do this, you lay the ground work for successful scripts.

Tasks

The next step, once you know what you are trying to script at the network level is to take a look at the tasks that scripting will take over for you. Some common tasks that lend themselves to scripting are things like software licensing monitoring, software updates and installations, collecting statistics on various network functions and resources, monitoring the usage of your network resources, and other such repetitive tasks. Besides the tasks themselves, you need to look at the frequency of the tasks. A print queue that hangs up once a year is probably not worth scripting, but the weekly web log processing is. Take a look at which tasks are cyclical and repetitive, and which ones happen at such random, or infrequent times as to be easier to do manually.

For the cyclical tasks, take a look at the periods involved. Don't just focus on how often the task needs to be done, but correlate that to the length of time the task takes. A daily, hour long task is a better candidate for scripting than a five minute task that runs every six months. Also, make sure you keep the priorities of the tasks in mind. While it may be fun to script an oddball job, making sure the critical ones are running when needed is usually more important.

Planning To Manage

The last section of the foundations of scripting a network is possibly more important than any of the others. You need to answer the question of whether or not a task *should* be scripted. Do you need to automate this task? Is it something that is properly suited to scripting? Do you need to have your fingers in that particular pie? Remember, you are doing this for a reason. You need to have a clear goal, a clear idea on what the desired outcome of your script, or scripts is going to be. Don't just start banging out scripts because you can.

Ask yourself, "Can this task be automated?" If the answer is yes, then decide on whether or not it should be. There are a lot of things that are done manually for non-technical reasons sometimes, and the folks you work with and for may not take kindly to you changing how things are done because it was Tuesday. Another question to ask is how mindless is this task. Face it, a lot of the things we do take the intelligence of a snail, and those things are often begging to be automated. (I have found that mindless usually holds repetitive's hand, so if a task is one, it's most likely the other too.) But, if the task needs the help of a roomful of theoretical physicists to run successfully, you probably don't want to script it.

If you are scripting data or statistics collections, then be *very* clear on what the results need to be. Collecting millions of data points automatically with the mother of all scripts is useless if you miss the one data point you really needed. Although this may seem almost insultingly simple advice, I've seen people go down the wrong road, and walked in my own footsteps on that road too many times. Make sure you know what you are trying to accomplish.

Applications

Don't Reinvent the Wheel

First of all, not everything can be scripted. Face it, there are tasks that will just have to be run manually. This is not a challenge to your ability as a scripter, just a fact of life. In any event, it's impossible to script things to the nth degree. Even if you came close, you would end up spending as much time maintaining the scripts as you did when you did everything manually. Wasted time is wasted time, regardless of how it happens. You have to balance time spent scripting and maintaining those scripts against the time you actually save. If you can't at least break even, figure out another way to do the task. Remember, time is money, and lots of it too.

Secondly, don't waste a lot of time searching down OSAXen, and memorizing the AppleScript Language Guide, in a quest to create a network management system in AppleScript. Play to AppleScript's strengths. It's a system level toolbox, that is at it's best when it is 'gluing' the abilities of one application or system feature to another, thereby giving both new capabilities. There are many excellent 'off the shelf' applications that will do a lot of what you would kill yourself trying to do manually. Use AppleScript with these applications, not instead of them. Not surprisingly, almost all network administration applications are scriptable. By taking advantage of the features in an existing program, you let the program and the computer do all the work, instead of you. Remember, you're an administrator, not a programmer. Programmers make tools, administrators use tools. Think like an administrator, and you'll do well.

Scriptable Applications For The Network

Here is a partial list of scriptable applications that I make frequent use of in my administration duties:

- **BBEdit** - <http://www.barebones.com>
  This is one of the best tools around for doing anything that requires searching and replacing of text. I know that you can do anything BBEdit does in Perl, but BBEdit is easier to use, and to script. I use it for everything from parsing LDIF files to reformatting email address book files. It's fast, easy to use, and can do almost anything you can think of with text, and it's very scriptable.

- **netOctopus** - <http://www.netopia.com/software/netoctopus/index.html>
  This product allows you to do configuration management, remote installs, software usage monitoring, collect statistics, almost anything a network administrator needs to do, and it works on Mac OS and Windows. It's also completely scriptable, and it allows you to script administration functions on Windows as well as the Mac OS. You haven't lived until you've done Windows Registry edits via AppleScript on 30 PCs at once. An amazing tool, and one I can't live without.

- **Timbuktu Pro** - <http://www.netopia.com/software/tb2/index.html>
  Timbuktu, also from Netopia, fills the one hole in netOctopus's arsenal, that of remote control. This application gives you total remote control over any Mac or Windows box, and like netOctopus, it's highly scriptable. This is one of those tools you never think you need until you try it.

- **Retrospect** - <http://www.dantz.com/>
  No treatise on networking is complete without a reference to Retrospect. Retrospect is one of the best backup applications on the market, and holds its own quite well against some very stiff competition. The nice thing about the scripting features in Retrospect is that Dantz makes it very easy to use it along with your email client of choice, so that you can have it email you status, instead of having to check things manually.

- **FunnelWeb Pro** - <http://www.activeconcepts.com>
  This is a web log analyzer, that will slice and dice your data any way you like, and build you some sweet little 3-D graphs of it all, so that you can easily show what your web servers are doing. A nicely scriptable application, that has a very nice feature set.

There are literally hundreds more applications like these, but to list them all would take me almost the entire magazine. Take a look at the applications you already use, chances are, most are scriptable.

Scripting

So, now that we have the basics of what to do before

you script, let's get into the fundamentals of actually scripting the network. These can be illustrated as What, Where, When, Why, and How.

What

The first thing to establish is what exactly is the code you are writing going to accomplish. Remember, you are automating a task, a single task in most cases. Keep your script focused on that task. While it may be fun to think about nifty features, that runs counter to what an administration script is about. In most cases, your script won't have an interface. This is either because you don't need one, or can't run the script if user input is required. The script will also need to be able to run unattended. Since the wee hours of the morning are the slow times on most networks, this is, coincidentally, the best time to run network scripts. Since you most likely don't like being at work at three in the morning, the script needs to be reliable as well as faceless.

For this type of scripting, 'the smaller the better' is not a bad mantra. From a reliability standpoint, two or three smaller scripts, each doing a single task are far better than a large script trying to handle those same two or three tasks. If it gets written, at some point it has to be debugged, and that gets harder exponentially with the size of the script. In general, if it's too big for the Script Editor, it's probably too big.

Where

Since you are going to be running many of these scripts from a single Mac, it behooves you to take some time to plan out your script server. As far as what kind of Mac to use, I would say that is dependant on the type and number of scripts running. From a speed to cost standpoint, I would say nothing less than a G3. Although there are a lot of server applications that run fine on older hardware, scripts are somewhat dependant on the CPU speed. I wouldn't necessarily run out and get a new G4 server, as there is not much, if any G4 – specific optimizations in AppleScript, so any speed increase would be due to overall system speed, as opposed to the Altivec units.

Since AppleScript is CPU – dependant, I would avoid servers that are already under heavy CPU use. In particular, avoid FileMaker Pro servers, (or any database server for that matter), or Retrospect servers. Both of those applications make heavy demands on virtually every system on the Mac, and neither reacts well to being in the background. In addition, if your script is in the background with either one of these applications, or applications of a similar type running, the CPU time the script gets may be so small as to cause timeout issues.

I have found that a not too heavily burdened AppleShareIP server is not a bad choice, as AppleShareIP is essentially a background – only server anyway, and does not make as much demand on CPU as it does on I/O. Make sure that you don't try to push an overloaded AppleShareIP server too hard though, as it will crash, which will do you no good whatsoever.

Obviously, the ideal situation is a standalone Mac. You want to have no earlier than MacOS 8.6 running on it, although 9.0.4, with the current AppleScript updates is a good combination as well. You will want to have some form of scheduling software on the server for your scripts. I have had good luck with iDO, but whichever one works best for you is as good a choice as any other. The nice thing about a script server is that with the exception of RAM, (64MB is just not enough), there is no need to update the stock hardware, so you can use a fairly low – cost configuration as the server.

When

Try to avoid scripts that can only run at specific times. This is not to say that there aren't times that are better to run scripts than others, but avoid scripts that can only run at ten in the morning on Wednesday. For example, trying to install Office 98 on fifteen Macs while you are running your backups is probably not the best example of timing, bandwidth – wise. On the other hand, if you have a lot of mobile users, you may have no choice. The trick here is to avoid that type of situation whenever possible. This is where my earlier admonition about knowing your network thoroughly comes in handy.

If at all possible, early in the morning is a great time. Even the biggest workaholic tends to knock off by three a.m. or so. This is an especially handy time if the script needs an hour or so of high bandwidth to run well. Make sure you are balancing your CPU and I/O capabilities with the script requirements. Don't try to make an hour script fit into a thirty minute time slot. In addition, if you have multiple scripts running, try not to shave the timings too close. The MacOS does a decent job of multitasking, but it does fall down here occasionally. Resist the temptation to see how many scripts you can have running at once, or kicking off within seconds of another one ending. Trying to push your script server too hard, or the timing program too hard is just begging for a painful example of Murphy's law.

Why

Although I talked about this before, it's important enough to repeat. Ask yourself again, *why* are you writing this script. Remember, you are doing this because you need to do it in a script, not because you can do it in a script. Assess your needs carefully here, and balance them against what the task requires. There are times when manual is better than automatic. The idea behind scripting your administrative duties is to make your life easier, not just as hard in a different way. If a job that normally takes five minutes once a month ends up needing a script that takes twenty minutes to run, a week to write, and is as twitchy as a cat in a room full of rocking chairs is not going to help you in the least. Make very, very sure that the end result of your scripting effort will be better than the current situation.

How

Once again I say plan, plan, plan. Since you are going to be affecting an entire network, you cannot afford to wing it. As

professional divers say, "Plan your dive, and dive your plan." In other words, take time, and figure out exactly what you need to do to get the results you want, then write the script to do exactly that, and no more. Of course, with actually writing the script, the tools you use become important.

For just writing the script, getting the code on media, I still prefer Apple's Script editor. It's very basic, and rather Spartan, but I personally find this to be an advantage, as it gets in my way less. Script Editor is a clean sheet of paper, and for some this can be a help. I also really like its dictionary display. If you want a clean display, but with more features, then Smile can be a help too. It's freeware, so if you find you don't like it, then you have only lost download time. It also has a devoted user community, who are more than happy to help you out in discovering Smile's hidden features.

As you get to a point in your script where you need better debugging capabilities then Smile or Script Editor give you, considering a higher end development environment, such as Scripter, or Script Debugger is not a bad idea. Both products do an excellent job, both are well-maintained, and will server you well. Like any tool, I recommend trying both out, and seeing which one fits your needs and your style best. I know that I have left out FaceSpan, which is another excellent development tool, but as that is more of an interface builder, and the idea here is to avoid user intervention, FaceSpan doesn't fit in.

Regardless of the tool used, there are some basic principles to follow. First of all, build the script gradually. Each script has a core function, which should be built first, and tested first. Once the core is done, then any other features that need to be there for the core to function correctly get built. Again, avoid 'feature-it is' like the plague here. It's unnecessary, and witll only cause you problems when you try to implement the script.

The second principle is don't reinvent the wheel unless you have to. As you write scripts, try to write them in a way that lets you use parts of them in other scripts easily. As you do this, you will find that you can build more of your new scripts from old scripts, saving you time on the actual coding phase. In line with this, keep up to date on various scripting resources, links to which can be found on Apple's AppleScript page, at http://www.apple.com/applescript/. There are a lot of examples, as guidebooks, and other resources that quite often have sample code that does a lot of what you want or need to do. As long as it is being given away, take it and use it. Make your life easier.

The final principle is 'Cool is nice, correct is better'. Remember, these scripts have to run day in and day out, without handholding. They need to run correctly at all times, and they need to run reliably at all times. You cannot afford to have a network script that crashes your script server, or other servers, or user machines. By keeping the scripts small and simple, you will find this easier to avoid.

Example Scripts

Now on to the fun part of all of this, actual AppleScripts. All of these are scripts I use in my work, most of them work with other applications, or from within other applications.

Enabling Windows NT 4 sp3 or greater to use plain text passwords

This is one I have had to use on all my Windows NT PCs. Since I have my application installation points on an AppleShareIP server, I need to turn on plain text passwords. By using AppleScript, and netOctopus, I can do this with ease. Besides, there's something fun about doing Windows Registry modifications from a Mac!

```
property NoSelectionErr : "There is no computer selected.
  Please select one!"

tell application "netOctopus"
  set theWindow to window "Computers"

  set theSelection to selection of theWindow
  if theSelection exists then
    add registry value of theSelection registry key path
"HKEY_LOCAL_MACHINE\\SYSTEM\\CurrentControlSet\\Services\\Rdr\
\Parameters" registry value name "EnablePlainTextPassword"
registry value type number value registry value "1"
  else
    display dialog NoSelectionErr buttons {"Okay"} with icon
stop
  end if
end tell
```


The first line sets up an error message which is there to tell me that I need to select a computer to run the scrip on. The next line sets the variable theWindow to the netOctopus window called 'Computers', which is a listing of all the computers, Mac and PC that netOctopus is currently managing. The third line sets theSelection to the computer selected in theWindow. This will be the computer that the script runs against. The if statement that follows makes sure that theSelection actually exists. If it doesn't, that means that no computer was selected, and the else clause is invoked, which displays our error message from the first line, and exits the script. If theSelection is valid, the line following the if statement comes into play. This line tells the netOctopus agent on the target PC to go down the registry path "HKEY_LOCAL_MACHINE\\SYSTEM\\CurrentControlSet\\Services\\Rdr\\Parameters" and add a new value to the key at that path. (The reason for the double '\' characters is that that character is both the file path delimiter for windows, and has special meaning in AppleScript. The double '\' allows that character to be passed as a text value. To see what actually is sent to the PC, just remove one '\' from the path.) The new value is named "EnablePlainTextPassword", is of type "number", and has a value of "1", indicating that it should be enabled. Once the script runs, the PC gets rebooted, and voila! Plaintext passwords. A very handy and useful script that is only twelve lines long.

Manually Scan Windows 9X drives with Norton AntiVirus

As anyone who has had to deal with 'Melissa', and her offspring are aware of, keeping ahead of virii is both difficult and critical. This script takes advantage of Norton AntiVirus's command-line scanner, and netOctopus's ability to run command-line PC programs to allow you to remotely scan drives on Windows machines. The script is very similar to the first one. In fact there are only two or three lines that differ, so I will only go over those lines.

```
property NoSelectionErr : "There is no computer selected.
  Please select one!"
property thePath : "C:\\Program Files\\Norton Antivirus"
property theExecutable : "navdx.exe"

tell application "netOctopus"
  set theWindow to window "Computers"

  set theSelection to selection of theWindow
  if theSelection exists then
    execute PC installer of theSelection executable path
thePath executable theExecutable execute using any drive
letter parameter "/L /B+ /M+ /HEUR:3 /S+ /REPAIR /DOALLFILES
/ZIPS"
  else
    display dialog NoSelectionErr buttons {"Okay"} with icon
stop
  end if
end tell
```

The second and third lines set up the path and application variables that are used in the script. The eighth line of the script is the one that does all the work. It tells netOctopus to run a PC installer on the selected computer. (This is actually the way to have netOctopus run *any* program that can be started from a command line, it doesn't have to be an installer.) The path and

executable are gotten from the variables thePath and theExecutable. The parameters to the executable are telling Norton Antivirus to scan all files on the drives, using maximum sensitivity, scanning inside of compressed files, and to repair any infected files that are found. A script that can save you a lot of trouble, and it is again, about twelve lines long.

Conclusion

There are a lot of scripts that I and others use daily, but I think I've made my point. By using AppleScript, and scriptable management applications, and taking advantage of the resources available from Apple and others, not only can you automate the tasks I demonstrated above, but everything from creating desktop printers and network configurations, to doing address book file conversion between email programs. I hope this article is a help, and I've included some links to various scripting resources in the references section.

References

- http://www.apple.com/applescript/
  This is the best place to start, as it has all of the Apple information on AppleScript, as well as all the links I have below.

- http://www.mainevent.com/
  The makers of Scripter, one of two professional – level AppleScript Development environment.

- http://www.lists.apple.com/lists.taf?function=subscribeinfo&listname=AppleScript-Users&digest=Yes&listType=Discussion
  This is where to sign up for the Apple AppleScript User's list. If you do almost any scripting, this is one of the best sources for help. Almost anyone who is anyone in the world of AppleScript is on this list, and the quality of help that I have received from it has always been absolutely top notch.

- http://www.AppleScriptSourcebook.com/
  Bill Cheeseman's site. If it isn't on Apple's site, then it's probably here. An absolutely fantastic resource.

- http://www.prefab.com/player.html
  The makers of PreFab Player, an application that allows you to use AppleScript with applications that either aren't scriptable, or have very poor scripting implementations.

- http://www.latenightsw.com/
  The makers of Script Debugger, the other professional level development environment for AppleScript.

- http://www.tandb.com/smile/
  A free AppleScript development environment. While it lacks the debugging and other features of Scripter and Script Debugger, it's a very capable product, and the price cannot be beat. MT

By Tim Monroe

The Informant

Getting and Setting Movie Information

INTRODUCTION

In the previous *QuickTime Toolkit* article, we saw how to create a QuickTime movie file that contains a single video track. We also learned a fair bit about the structure of QuickTime movies (as collections of tracks) and QuickTime movie files (as collections of atoms). In this article, we'll continue on with the general topic of creating and configuring QuickTime movie files. We'll see how to get various pieces of information about QuickTime movies and movie files; we'll also see how to add information to a QuickTime movie to help the user determine what's in a movie.

To get an idea of what we're going to accomplish here, let's suppose that we're running some version of the MoviePlayer application (the predecessor to the current QuickTime Player application). MoviePlayer's Movie menu contains the item "Show Copyright...". If we select that item immediately after having opened the movie file we created in the previous article, we'll see the *movie information dialog box* shown in **Figure 1**.

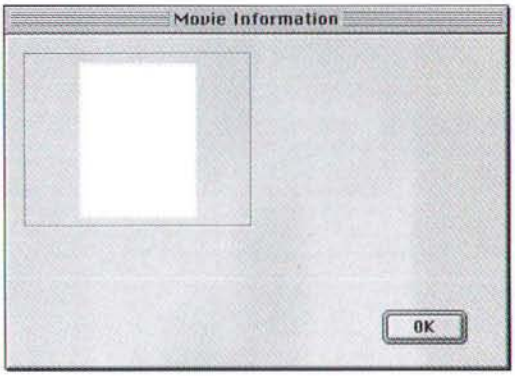


Figure 1: *The movie information dialog box for our new movie*

As you can see, this is not particularly helpful. The only real "information" visible to the user is the first frame of the movie, which happens to be a solid white rectangle. It would be better to display some other frame of the movie and to add some descriptive information to the other panes of the dialog box. **Figure 2** shows a much more useful movie information dialog box.

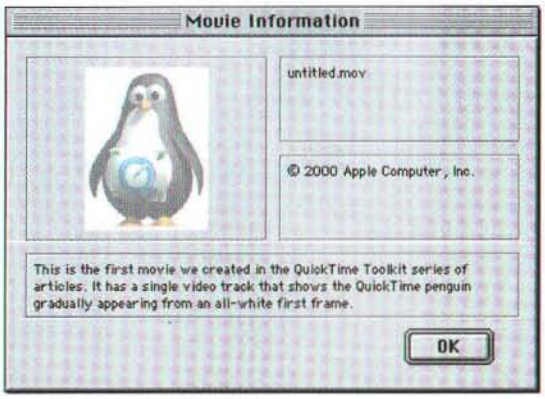
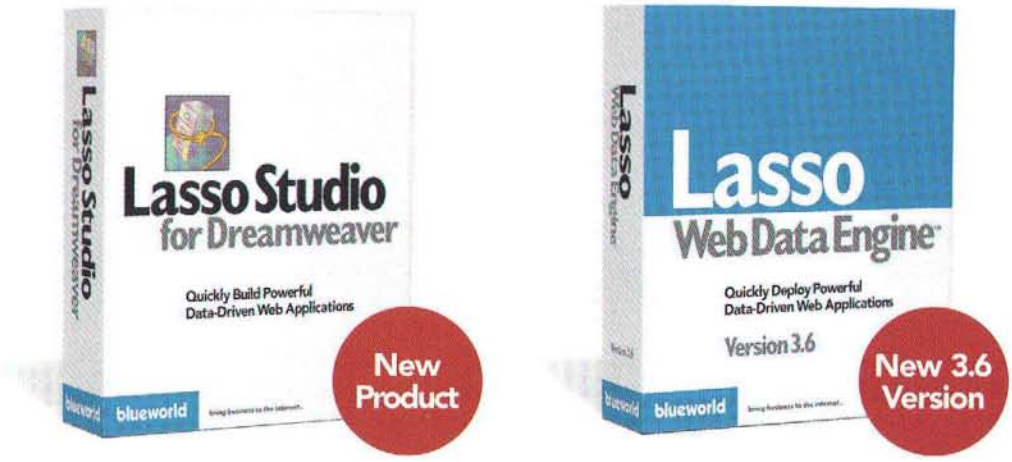


Figure 2: *The revised movie information dialog box for our new movie*

Tim Monroe has worked at Apple for over 10 years, first as a technical writer in the Inside Macintosh group and later as a software engineer in the QuickTime group. Currently he is developing sample code and utilities for the QuickTime software development kit. You can reach him at monroe@apple.com.

Quickly Build and Deploy Powerful Data-Driven Web Applications



Lasso Studio and Lasso Web Data Engine™ Lead The Way.

Building custom and shrink-wrapped database-driven Web applications requires a whole new way of doing things. Having pioneered the Web Data Engine™ over three years ago, Blue World and the Lasso Web Data Engine consistently lead the way providing a feature set Web developers describe as "incredible." Build online stores, discussion forums, resource management systems and other demanding database-driven Web applications with unrivaled performance, ease, security, extensibility, control and flexibility. Develop using multiple languages—including LDML, CDML, Server-Side JavaScript, Java and XML— and deploy across multiple platforms. Your Lasso code works identically regardless to which database you're connected. Lasso solutions for FileMaker® Pro databases easily scale to big iron ODBC-compliant databases like Oracle, Informix, Sybase and more with little or no change. What's more, the Lasso Java Application Programming Interface (LJAPI) provides developers an easy-to-use Java-based API for unprecedented extensibility.

Find out why hundreds of thousands of websites rely on award-winning Lasso technology for their business critical Web data. Download a 30-day evaluation copy at www.blueworld.com/download/ or order securely online today at the Blue World Store at store.blueworld.com.

Lasso Product Line – The leading Web tools for Macintosh and beyond.

bring business to the internet.™

blueworld

Part of our task here will be to see how to modify the movie file so that selecting "Show Copyright..." displays the dialog box in Figure 2 rather than the one in Figure 1. In a nutshell, this involves setting the movie poster to some frame other than the first frame, which is the default poster frame; it also involves attaching three new pieces of movie user data to the movie file. Along the way, we'll also learn how to set the preview that is contained in the file-opening dialog boxes displayed by calls to the StandardGetFilePreview and NavGetFile functions. **Figure 3** shows a typical file-opening dialog box with a preview.

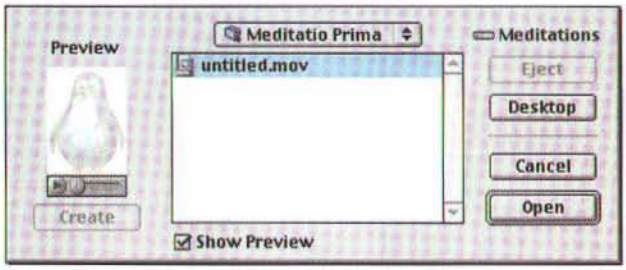


Figure 3: *A preview contained in a file-opening dialog box*

Our sample application this time around is called QTInfo. As usual, it's based directly on the QTShell sample application that we've developed previously. QTInfo is just QTShell with one additional source code file (QTInfo.c) and some additional resources. **Figure 4** shows the Test menu supported by QTInfo.

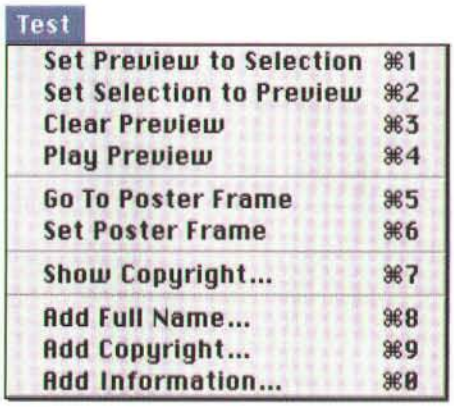


Figure 4: *The Test menu in QTInfo*

As you can see, QTInfo provides the "Show Copyright..." menu item, as well as a number of other items that allow us to get and set various kinds of movie information. It turns out that we can handle the "Show Copyright..." item with a single line of code:

```
ShowMovieInformation(myMovie, gModalFilterUPP, 0L);
```

The ShowMovieInformation function was introduced in QuickTime version 2.0, but has (to my knowledge) never been documented. ShowMovieInformation simply displays the movie information dialog box, which includes the movie poster image, the name of the movie, the movie's copyright information, and some other information. If you pass a universal procedure pointer to a modal dialog event filter function in the second parameter, you'll get a movable modal dialog box; otherwise, you'll get a standard non-movable modal dialog box, as shown in **Figure 5**.

Figure 5: *A non-movable movie information dialog box*

Movie Posters

A *movie poster image* (or, more briefly, a *movie poster*) is a single image that represents a QuickTime movie. The images in the top-left panes of **Figures 1**, **2**, and **5** are movie posters, suitably resized to fit into the available space in the movie information dialog box. A movie poster is defined by specifying a *movie poster time* and one or more *movie poster tracks*. The movie poster time specifies the time in the movie at which the image is to found, and the movie poster tracks specify which tracks in the movie are to be used to create the movie poster. Typically a single track is used as the movie poster track, but in theory two or more video tracks (or other tracks with visible data) could contribute to the final movie poster image. If a movie has no track designated as a movie poster track, then the movie won't have a poster, no matter what the movie poster time is set to. Let's see how to work with poster times and tracks.

Getting and Setting Movie Poster Times

The default movie poster time is 0, which picks out the first frame in the movie. As we saw earlier, it's sometimes useful to designate some other time as the movie poster time. The function QTInfo_SetPosterToFrame, defined in Listing 1, sets the currently-displayed movie frame to be the movie poster image. (QTInfo calls QTInfo_SetPosterToFrame in response to the "Set Poster Frame" menu item.)

Listing 1: Setting the movie poster time to the current movie time

QTInfo_SetPosterToFrame

```
OSErr QTInfo_SetPosterToFrame
  (Movie theMovie,

MovieController theMC)
{
  TimeValue          myTime;
  ComponentResult    myErr = noErr;

  // stop the movie from playing
  myErr = MCDoAction(theMC,
mcActionPlay, (void *)0L);
  if (myErr != noErr)
    goto bail;

  myTime = GetMovieTime
    (theMovie, NULL);
  SetMoviePosterTime
    (theMovie, myTime);

  myErr = MCMovieChanged
    (theMC, theMovie);

bail:
  return((OSErr)myErr);
}
```

As you can see, QTInfo_SetPosterToFrame first calls MCDoAction to set the movie play rate to 0, which effectively stops the movie from playing. (If the movie is already stopped, this call has no effect.) Then QTInfo_SetPosterToFrame retrieves the current movie time by calling the GetMovieTime function and sets the movie poster time to the current movie time by calling the SetMoviePosterTime function.

QTInfo_SetPosterToFrame finishes up by calling the MCMovieChanged function, which informs the movie controller that we've made changes to the movie using the Movie Toolbox. As we've seen in past articles, there are often two ways to change some characteristic of a movie: using Movie Toolbox functions and using movie controller functions. When a movie is associated with a movie controller and when we make a change to the movie using the Movie Toolbox, it's usually necessary to keep things in sync by calling MCMovieChanged. For example, if we change the size of the movie by calling SetMovieBox, we'd need to call MCMovieChanged so that the movie controller can update itself appropriately.

In the present case, there is no movie controller action to set the poster frame, so we used the Movie Toolbox function SetMoviePosterTime. Then we called

MCMovieChanged on the offhand chance that the movie controller might actually care about the poster time. I've tried running QTInfo without the call to MCMovieChanged here and no harm appears to result, but it's better to be safe than sorry. As a general rule, if you have a movie controller associated with a movie and you use the Movie Toolbox to effect some change in the movie, call MCMovieChanged to inform the movie controller of the change.

QTInfo supports the "Go To Poster Frame" menu item, which sets the current movie time to the movie poster time. Listing 2 defines the QTInfo_GoToPosterFrame function, which does just that.

Listing 2: Setting the current movie time to the movie poster time

QTInfo_GoToPosterFrame

```
OSErr QTInfo_GoToPosterFrame (Movie theMovie,
                              MovieController theMC)
{
  TimeRecord          myTimeRecord;
  ComponentResult     myErr = noErr;

  // stop the movie from playing
  myErr = MCDoAction(theMC, mcActionPlay, (void *)0L);
  if (myErr != noErr)
    goto bail;

  // set up a time record with the desired movie time, scale, and base
  myTimeRecord.value.hi = 0;
  myTimeRecord.value.lo =
                        GetMoviePosterTime(theMovie);
  myTimeRecord.base = GetMovieTimeBase(theMovie);
  myTimeRecord.scale = GetMovieTimeScale(theMovie);

  myErr = MCDoAction(theMC, mcActionGoToTime,
&myTimeRecord);

bail:
  return((OSErr)myErr);
}
```

In this case, there *is* a movie controller action that we can use to set the current movie time, namely mcActionGoToTime. As a result, there is no need to call MCMovieChanged after we've made this change to the movie (since we made the change using movie controller actions, not the Movie Toolbox). We did of course use Movie Toolbox functions to get information needed to fill in the TimeRecord structure whose address we pass to MCDoAction, but those functions didn't make any changes to the movie; they simply gave us information about the movie.

Working with Movie Poster Tracks

I mentioned earlier that a movie's poster image is determined both by the movie poster time and by the movie poster tracks. Each track in a movie has a *track usage*, which indicates whether the track is used in the movie, the movie poster, the movie preview, or any combination of these. For instance, a movie can include a video track that consists of a single frame, and that track can be the only one in the movie that is used in the movie poster. In this way, it's possible to have a movie poster that is not simply one of the frames in the movie, but is some other image altogether.

We can query a track's usage by calling the GetTrackUsage function. GetTrackUsage returns a long integer whose bits encode the track usage. Currently, these three bits are defined:

```
enum {
  trackUsageInMovie           = 1 << 1,
  trackUsageInPreview         = 1 << 2,
  trackUsageInPoster          = 1 << 3
};
```

By default, a track can be used in any of these three ways, so calling GetTrackUsage on most tracks will return a value of 0x0000000E (that is, binary 1110). But we can change this default setting by calling SetTrackUsage, passing it a long integer that has the appropriate flags set or clear. We'll see some calls to GetTrackUsage and SetTrackUsage in a moment. For now, it's important to understand that a track usage value indicates a track's potential use, not its actual use. That is to say, if a particular track has a track usage value with the trackUsageInPoster flag set, the poster image might not actually include any data from that track. This might happen if the movie poster time is set to a time at which that track has no data (perhaps the track offset is greater than the movie poster time). Similarly, a track's usage value can have the trackUsageInPreview flag set, even if the movie has no preview. To repeat, the track usage determines the uses a track *can have*, not the uses it *actually has*.

Let's see how this works in practice. When QTInfo wants to adjust the state of its menus, it needs to know whether the movie in the frontmost window has a poster image. If there is no poster image, then it should disable the "Go To Poster Frame" menu item. To determine whether a movie has a poster image, QTInfo calls the QTInfo_MovieHasPoster function defined in Listing 3. Essentially, QTInfo_MovieHasPoster looks at each track in the movie, retrieves the track's usage value, and checks to see whether the trackUsageInPoster flag is set in that value. If there is at least one track that is capable of contributing data to the movie poster image, we'll happily count the movie as having a poster image.

Listing 3: Determining whether a movie has a poster image

QTInfo_MovieHasPoster

```
Boolean QTInfo_MovieHasPoster (Movie theMovie)
{
  long            myCount = 0L;
  long            myIndex = 0L;
  Track          myTrack = NULL;
  long            myUsage = 0L;
  Boolean        myHasPoster = true;

  // make sure that some track is used in the movie poster
  myCount = GetMovieTrackCount(theMovie);
  for (myIndex = 1; myIndex <= myCount; myIndex++) {
```

```
    myTrack = GetMovieIndTrack(theMovie, myIndex);
    if (myTrack == NULL)
      continue;

    myUsage = GetTrackUsage(myTrack);
    if (myUsage & trackUsageInPoster)
      break;
  // we found a track with the trackUsageInPoster flag set; break out of the loop
  }

  if (myIndex > myCount)
    myHasPoster = false;
          // we went thru all tracks without finding one with a poster usage

  return(myHasPoster);
}
```

The QTInfo_MovieHasPoster function is instructive for other reasons as well, in particular because it shows how to iterate through all tracks in a movie. As you can see, it begins by calling the GetMovieTrackCount function to determine how many tracks the specified movie contains. Then it repeatedly calls the GetMovieIndTrack function to get a track identifier for each of those tracks. The Movie Toolbox also supplies the GetMovieIndTrackType function, which allows us to iterate through all tracks of a specific type (say, all video tracks). We won't have occasion to use GetMovieIndTrackType in this article, but we will in the future.

Movie Previews

A *movie preview* is a short, dynamic representation of a QuickTime movie. Typically, a movie preview is an excerpt of the movie itself (for example, the first few seconds of the movie). But, like a movie poster, a movie preview can consist of data that is not used in the normal playback of the movie. Once again, the usage values of the tracks in the movie determine the actual contents of the movie preview.

Defining Movie Previews

We specify a movie preview by giving its start time, its duration, and its tracks. The recommended duration is about 3 to 5 seconds, but you are free to use a longer or shorter duration if you wish. An easy way to let the user specify a movie preview is to provide the "Set Preview to Selection" menu item, which uses the start time and duration of the current movie selection as the start time and duration of the movie preview. Listing 4 shows how to set the movie preview to the current movie selection.

Listing 4: Setting the movie preview to the current movie selection

QTInfo_SetPreviewToSelection

```
OSErr QTInfo_SetPreviewToSelection (Movie theMovie,
                                    MovieController
theMC)
{
  TimeValue         myStart;
  TimeValue         myDuration;
  ComponentResult     myErr = noErr;

  GetMovieSelection(theMovie, &myStart, &myDuration);
  SetMoviePreviewTime(theMovie, myStart, myDuration);

  myErr = MCMovieChanged(theMC, theMovie);

  return((OSErr)myErr);
}
```

The QTInfo_SetPreviewToSelection function is simplicity itself. We just call GetMovieSelection to get the current movie start time and duration, and then we pass those same values to the SetMoviePreviewTime function. We need to call MCMovieChanged here because we changed the characteristics of the movie (in particular, its movie preview) using the Movie Toolbox.

As we've seen, QTInfo also provides the "Set Selection to Preview" menu item, which sets the movie's selection to the current movie preview. Listing 5 defines the function QTInfo_SetSelectionToPreview, which performs this operation.

Listing 5: Setting the current movie selection to the movie preview

QTInfo_SetSelectionToPreview

```
OSErr QTInfo_SetSelectionToPreview (Movie theMovie,
                                    MovieController theMC)
{
  TimeValue         myStart;
  TimeValue         myDuration;
  ComponentResult     myErr = noErr;
```

```
  GetMoviePreviewTime(theMovie, &myStart, &myDuration);
  SetMovieSelection(theMovie, myStart, myDuration);

  myErr = MCMovieChanged(theMC, theMovie);

  return((OSErr)myErr);
}
```

We need to enable or disable the "Set Preview to Selection" and "Set Selection to Preview" menu items, depending on whether a movie has a selection or preview. It's easy to determine whether a movie has a selection: we can simply call GetMovieSelection and check to see whether the duration returned to us is greater than 0, like this:

```
GetMovieSelection(myMovie, &myStart, &myDuration);
myHasSelection = (myDuration > 0);
```

But it's a bit more complicated to determine whether a movie has a movie preview. We need to check to see both whether the movie has a non-zero movie preview duration and whether any tracks in the movie are used in the movie preview. Listing 6 defines the QTInfo_MovieHasPreview function, which performs both of these checks. As you can see, QTInfo_MovieHasPreview is very similar to QTInfo_MovieHasPoster (Listing 3).

Listing 6: Determining whether a movie has a preview.

QTInfo_MovieHasPreview

```
Boolean QTInfo_MovieHasPreview (Movie theMovie)
{
  TimeValue       myStart;
  TimeValue       myDuration;
  long              myCount = 0L;
  long              myIndex = 0L;
  Track           myTrack = NULL;
  long              myUsage = 0L;
  Boolean         myHasPreview = false;

  // see if the movie has a positive preview duration
  GetMoviePreviewTime(theMovie, &myStart, &myDuration);
  if (myDuration > 0)
    myHasPreview = true;

  // make sure that some track is used in the movie preview
  myCount = GetMovieTrackCount(theMovie);
  for (myIndex = 1; myIndex <= myCount; myIndex++) {
    myTrack = GetMovieIndTrack(theMovie, myIndex);
    if (myTrack == NULL)
      continue;

    myUsage = GetTrackUsage(myTrack);
    if (myUsage & trackUsageInPreview)
      break;
// we found a track with the trackUsageInPreview flag set; break out of the loop
  }

  if (myIndex > myCount)
    myHasPreview = false;
        // we went thru all tracks without finding one with a preview usage

  return(myHasPreview);
}
```

Playing Movie Previews

The Movie Toolbox provides an easy way to show the user the exact contents of a movie preview. We can call the PlayMoviePreview function, like this:

```
PlayMoviePreview(myMovie, NULL, 0L);
```

When we execute PlayMoviePreview, the Movie Toolbox puts our movie into preview mode, plays the movie preview in the movie's graphics port, and then sets the movie back into normal playback mode. When the movie returns to normal playback mode, the current movie time is set to the end of the movie preview.

The second parameter to PlayMoviePreview is a universal procedure pointer to a *movie callout function*, which the Movie Toolbox calls repeatedly while the preview is playing. We might use a movie callout function to provide a way for the user to stop the preview from playing (perhaps by checking the event queue for some particular key press). If we don't use a movie callout function, then the call to PlayMoviePreview is essentially synchronous: no other events will be processed until the movie preview finishes playing.

The Movie Toolbox provides a way to play a movie preview without blocking other processing. We can call SetMoviePreviewMode with its second parameter set to true to put a particular movie into preview mode. SetMoviePreviewMode restricts the active segment of the movie to the segment of the movie picked out by the preview's start time and duration; it also restricts the active tracks to those that have the trackUsageInPreview flag set in their track usage values. Once a movie has been set into preview mode, we can start it and stop it by calling the StartMovie and StopMovie functions. To exit movie preview mode, we can call SetMoviePreviewMode with its second parameter set to false. (Note that QTInfo does not illustrate this method of playing movie previews; it calls PlayMoviePreview.)

Clearing Movie Previews

Sometimes it's useful to clear a movie preview from a movie. We can do this by setting both the start time and duration of the movie preview to 0, like this:

```
SetMoviePreviewTime(theMovie, 0, 0);
```

Executing this line alone effectively prevents any movie preview from being displayed. But we also want to perform a few other actions. For one thing, we should remove any tracks from the movie that are used *only* in the movie preview. We can do this by examining the track usage value for each track in the movie and, if the usage value indicates that a track is used in the movie preview but not in the movie or the movie poster, calling DisposeMovieTrack to remove the track from the movie.

We're giving you four days in October to catch up to the future.

Announcing QuickTime Live! October 9-12 at the Beverly Hilton Hotel. With over 50 million downloads, QuickTime™ is fast becoming the industry standard for multimedia content on the Mac® OS and Windows. At QuickTime Live! we're presenting four days of exhibits, workshops and conferences to help you put QuickTime to work brilliantly–from CD content to web streaming. If you're a multimedia pro, it's your future. Don't miss it. For details and registration information, visit our website, www.apple.com/quicktime-

Also, once we've removed any tracks that were used only in the movie preview, we should go back through the remaining tracks and reset their track usage values so that they can be used as part of a movie preview, if one is subsequently added. If we don't do this, the user might be unable to create a new movie preview, since it's possible that none of the remaining tracks in the movie has the trackUsageInPreview flag set in its track usage value.

Listing 7 defines the QTInfo_ClearPreview function, which performs all three of these actions.

Listing 7: Clearing a movie preview

QTInfo_ClearPreview

```
OSErr QTInfo_ClearPreview (Movie theMovie,
                           MovieController theMC)
{
  long                myCount = 0L;
  long                myIndex = 0L;
  Track               myTrack = NULL;
  long                myUsage = 0L;
  ComponentResult     myErr = noErr;

  // set the movie preview start time and duration to 0
  SetMoviePreviewTime(theMovie, 0, 0);

  // remove all tracks that are used *only* in the movie preview
  myCount = GetMovieTrackCount(theMovie);
  for (myIndex = myCount; myIndex >= 1; myIndex--) {
    myTrack = GetMovieIndTrack(theMovie, myIndex);
    if (myTrack == NULL)
      continue;

    myUsage = GetTrackUsage(myTrack);
    myUsage &= trackUsageInMovie | trackUsageInPreview
                                 | trackUsageInPoster;
    if (myUsage == trackUsageInPreview)
      DisposeMovieTrack(myTrack);
  }

  // add trackUsageInPreview to any remaining tracks that are in the movie
  // (so that subsequently setting the preview to a selection will include
  // these tracks)
  myCount = GetMovieTrackCount(theMovie);
  for (myIndex = 1; myIndex <= myCount; myIndex++) {
    myTrack = GetMovieIndTrack(theMovie, myIndex);
    if (myTrack == NULL)
      continue;

    myUsage = GetTrackUsage(myTrack);
    if (myUsage & trackUsageInMovie)
      SetTrackUsage(myTrack, myUsage |
                             trackUsageInPreview);
  }

  myErr = MCMovieChanged(theMC, theMovie);

  return((OSErr)myErr);
}
```

File Previews

Now consider this question: when we call StandardGetFilePreview (or NavGetFile with the preview pane enabled), what is displayed in the preview section of the file-opening dialog box? Before you answer, take a look back at **Figure 3**. I suspect you're inclined to say that it's the movie preview. But before you make that your final answer, take a look at **Figure 6**, which shows another file-opening dialog box.

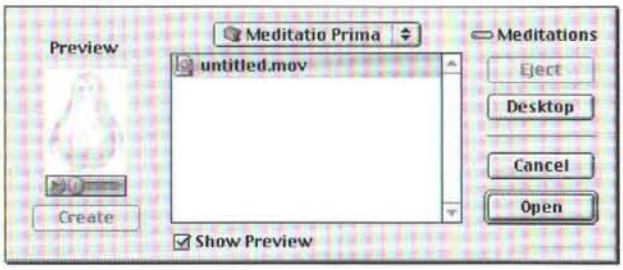


Figure 6: *A poster contained in a file-opening dialog box*

And then take a look at **Figure 7**, which shows yet another file-opening dialog box.

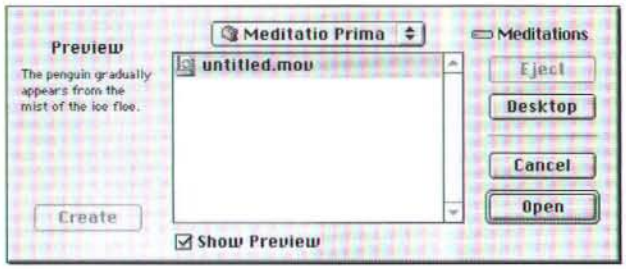


Figure 7: *A description contained in a file-opening dialog box*

Thoroughly confused? I thought so.

The correct answer to our little quiz is that the preview displayed in a file-opening dialog box is what's called a *file preview*, which is any information that gives the user an idea of what's in the file. As we've seen, the file preview can be a movie poster or a movie preview (if the file is a movie file) *or any other data that describes or represents the file.* On Macintosh computers, the default file preview for a QuickTime movie file is a miniature version of the movie poster frame, while on Windows computers it's the first 10 seconds of the movie. But we are free to specify some other information as the file preview, if we so desire. Let's see how file previews are stored and created, to make this all perhaps a bit clearer.

Accessing File Previews

On Macintosh computers, when StandardGetFilePreview or NavGetFile needs to display a file preview for a QuickTime movie file, it first checks to see whether the file is a double-fork or single-fork movie file. If the selected file is a double-fork movie file, StandardGetFilePreview or NavGetFile looks in the resource fork for a resource of type 'pnot'. The data in a 'pnot' resource is organized as a *preview resource record*, which is defined in ImageCompression.h like this:

```
struct PreviewResourceRecord {
  unsigned long                modDate;
  short                        version;
  OSType                       resType;
  short                        resID;
};
```

The resType and resID fields specify the type and ID of some other resource, which contains the actual file preview data or which itself indicates where to find that data. (Let's call that other resource the *preview data resource*.) For instance, if resType and resID pick out a resource of type 'PICT', then the picture in that resource will be used as the file preview (as in **Figure 6**). Similarly, if resType and resID pick out a resource of type 'TEXT', then the text in that resource will be used as the file preview (as in **Figure 7**). If resType and resID pick out a resource of type 'moov', then the movie preview start time and duration specified in that resource will be used to pick out the file preview (as in **Figure 3**). If there is no movie resource in the resource fork, then resID should be set to –1 (0xFFFF), which tells StandardGetFilePreview to use the movie preview whose start time and duration are stored in the movie atom in the file's data fork.

In single-fork movie files, there is no resource fork. So StandardGetFilePreview opens the data fork and looks for an atom of type 'pnot', which it interprets in the same way as a 'pnot' resource, with one small difference: the resID field is interpreted as a 1-based index of atom types in the movie file. For example, if the resType field in the 'pnot' atom in a single-fork movie file is 'PICT' and the resID field is 1, then StandardGetFilePreview looks for the first atom in that file of type 'PICT', which it then uses as the file preview.

There are a couple of "gotchas" here that you should be aware of. First, the NavGetFile function currently seems to work only with file previews specified by 'pnot' resources. If you're creating single-fork movie files (as I have recommended), don't expect them to have file previews in the file-opening dialog boxes displayed by NavGetFile. Worse yet, NavGetFile doesn't seem to know how to handle movie previews as file previews, even in double-fork movie files. Finally, StandardGetFilePreview doesn't seem to know how to handle movie previews as file previews when stored in single-fork movies. (At least, I haven't been able to get them to work.) Our strategy below will be to create single-fork movie files with 'pnot' atoms in the format that is publicly documented. Then we'll just have to wait until StandardGetFilePreview and NavGetFile to catch up to us (as I expect they will).

(By the way, you might be wondering why file preview resources and atoms have the type 'pnot'. The 'p' of course is for "preview'; but what's the 'not' all about? The constant assigned to the component that displays file previews is of no help in deciphering this:

```
enum {
  ShowFilePreviewComponentType    =
FOUR_CHAR_CODE('pnot')
};
```

I'm told, by a knowledgeable source, that early versions of the QuickTime software — prior to version 1.0 — contained a preview component that wasn't very good. When the replacement was written, it was given the type 'pnot' as an abbreviation for "Preview? Not!")

Creating File Previews

Ideally, we'd like the QuickTime movie files that we create to have file previews, so that the user can get a reasonable idea of what's in those files when they appear in the list of files in the file-opening dialog box. The Image Compression Manager provides the MakeFilePreview function, which we can use to create file previews. *Inside Macintosh* recommends calling MakeFilePreview whenever we save a movie file. So we can insert a call to MakeFilePreview in the two functions QTFrame_UpdateMovieFile and QTFrame_SaveAsMovieFile (both in the file ComFramework.c) which handle the "Save" and "Save As" menu commands:

```
MakeFilePreview(myRefNum, (ICMProgressProcRecordPtr)-1);
```

MakeFilePreview sets the file preview for the file specified by the myRefNum parameter to be the current movie preview, if one exists; if the movie does not have a movie preview, then MakeFilePreview creates a thumbnail version of the movie poster image and sets it to be the file preview. (A *thumbnail* is a small copy of an image, typically 80 pixels on the longer side.) If we want to create a file preview using some other type of data (for instance, text data), we can call the ICM function AddFilePreview, which allows us to specify the type of data we want to use.

But there is one big problem here: MakeFilePreview and AddFilePreview always add the file preview information to the movie file's resource fork. Indeed, MakeFilePreview and AddFilePreview will go so far as to *create* a resource fork for the movie file if it doesn't already have one, so that they have a place to put the file preview they create. Needless to say, this behavior is going to wreak havoc with our desire to create only single-fork movie files. So, however tempting it might be to use MakeFilePreview to create our file previews, we're just going to have to resist that temptation.

In short, QuickTime does not currently provide any API to add a file preview to a single-fork movie file. But based on what we learned above about the way file previews are stored in single-fork files and on what we learned in the previous article about the general structure of QuickTime movie files, it won't be too hard for us to do this ourselves. For, we know that a single-fork movie file is just a collection of atoms. And a file preview can be stored in a single-fork movie as an atom of type 'pnot' together with a preview data atom that holds the actual preview data. So all we really need to do is append an atom or two to a single-fork movie file. **Figure 8** shows a single-fork movie file with no file preview (top) and that same file with a file preview (bottom). We'll define a function QTInfo_MakeFilePreview that we can use to turn the top file into the bottom file.

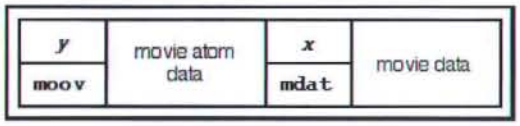


Figure 8: *A single-fork movie file before and after adding a file preview*

QTInfo_MakeFilePreview is declared like this:

```
OSErr QTInfo_MakeFilePreview (Movie theMovie,
  short theRefNum, ICMProgressProcRecordPtr
theProgressProc)
```

As you can see, QTInfo_MakeFilePreview has the same parameters as MakeFilePreview, except that we also pass the movie identifier to QTInfo_MakeFilePreview. The second parameter to QTInfo_MakeFilePreview is the file reference number of the open movie file. If QTInfo_MakeFilePreview is passed a reference to a resource fork, then it can just call MakeFilePreview to add the required preview resources to that resource fork, like this:

```
if (QTInfo_IsRefNumOfResourceFork(theRefNum)) {
  myErr = MakeFilePreview(theRefNum, theProgressProc);
  goto bail;
}
```

But if QTInfo_MakeFilePreview is passed the file reference number of a data fork, then we'll assume that we must add the file preview information to the data fork. This involves adding a 'pnot' atom to the data fork, as well as a preview data atom. Recall that an atom consists of an atom header and some atom data. For a 'pnot' atom, the atom data is a record of type PreviewResourceRecord. So we can construct the 'pnot' atom like this:

```
PreviewResourceRecord   myPNOTRecord;
unsigned long           myAtomHeader[2];  // an atom header

// fill in the 'pnot' atom header
myAtomHeader[0] = EndianU32_NtoB(sizeof(myAtomHeader) +
                                 sizeof(myPNOTRecord));
myAtomHeader[1] =

EndianU32_NtoB(ShowFilePreviewComponentType);

// fill in the 'pnot' atom data
GetDateTime(&myModDate);
myPNOTRecord.modDate = EndianU32_NtoB(myModDate);
myPNOTRecord.version = EndianS16_NtoB(0);
myPNOTRecord.resType = EndianU32_NtoB(myPreviewType);
myPNOTRecord.resID = EndianS16_NtoB(1);
```

All data in predefined QuickTime movie atoms must be in big-endian format, so here we use the macros EndianU32_NtoB and EndianS16_NtoB to convert from the computer's native-endian format into big-endian format.

Notice that the resType field is set to myPreviewType. We'll create a file preview that is either a movie preview or a movie poster thumbnail, depending on whether the movie has a movie preview:

```
If (QTInfo_MovieHasPreview(theMovie))
  myPreviewType = MovieAID;
else
  myPreviewType = kQTFileTypePicture;
```

The next thing we need to do is write the 'pnot' atom data onto the end of the movie file. We can use the File Manager functions GetEOF, SetEOF, SetFPos, and FSWrite to do this. See Listing 8 below for the exact steps involved in writing the data into the file.

Now we need to write the actual preview data into an atom of the appropriate type. For a movie preview, we can just point to the 'moov' atom, which contains the start time and duration of the movie preview. For a file preview that contains a thumbnail of the movie poster frame, we need to retrieve the movie poster frame, create a thumbnail image from it, and write the atom onto the end of the movie file. We can call GetMoviePosterPict to get the movie poster image:

```
myPicture = GetMoviePosterPict(theMovie);
```

Then we can call the ICM function MakeThumbnailFromPicture to reduce the poster image to a thumbnail image:

```
myThumbnail = (PicHandle)NewHandleClear(4);
myErr = MakeThumbnailFromPicture(myPicture, 0,
myThumbnail,
                                 theProgressProc);
```

If MakeThumbnailFromPicture successfully creates the thumbnail image, we need to fill in an atom header and write

// Like most Mac developers,

// I easily spend 12 hours a day

// staring at line after line of C++ code in tiny, 9 point Monaco.
// Sometimes it makes my eyes feel like they're on fire.

{
So the last thing I need is some
fuzzy monitor that adds to my headaches.
}

/****** begin excitement ******/

// That's why I'm so jazzed about this SGI monitor.

// Its ultra-high resolution = 1600 x 1024 and dpi = 110,
// giving me razor-sharp contrast.
// And the high refresh rate is
// perfect for poring through lines of code.

// At first, I was amazed at the clarity, the fine details that emerged.

{
It was like seeing things for the first time.
}

// Later, though, I learned to appreciate the wide aspect ratio (= 16:10),
// with a generous 17.3 inches of viewing area. SGI's 1600SW
// lets me have all my documents viewable at once, and it's
// a flat panel so it fits on my desk with room to spare.

// From the moment I saw this thing I was hooked. You will be, too.

{
Especially when you find out
how affordable it is.
}

// Check it out.

/****** end excitement ******/

Michael Whittingham
V.P., Engineering
Wired, Inc.

www.sgi.com/flatpanel

This advertisement was written by an actual engineer. This is the first time in 7 months that he's been seen during the day.

sgi™

the header and thumbnail data into the movie file as an atom of type 'PICT', like this:

```
myAtomHeader[0] = EndianU32_NtoB(sizeof(myAtomHeader) +
                  GetHandleSize((Handle)myThumbnail));
myAtomHeader[1] = EndianU32_NtoB(myPreviewType);

// write the atom header into the file
mySize = sizeof(myAtomHeader);
myErr = FSWrite(theRefNum, &mySize, myAtomHeader);
if (myErr == noErr) {
  // write the atom data into the file
  mySize = GetHandleSize((Handle)myThumbnail);
  myErr = FSWrite(theRefNum, &mySize, *myThumbnail);
}
```

Listing 8 brings all of this together into a single function that writes the appropriate file preview into the resource fork or the data fork, depending on the kind of file reference number passed to it in the second parameter.

Listing 8: Creating a file preview

QTInfo_MakeFilePreview

```
OSErr QTInfo_MakeFilePreview (Movie theMovie,
  short theRefNum, ICMProgressProcRecordPtr
theProgressProc)
{
  unsigned long           myModDate;
  PreviewResourceRecord   myPNOTRecord;
  long                    myEOF;
  long                    mySize;
  unsigned long           myAtomHeader[2];// an atom header
  OSType                  myPreviewType;
  OSErr                   myErr = noErr;

  // determine whether theRefNum is a file reference number of a data fork
or
  // a resource fork; if it's a resource fork, then we'll just call the existing
ICM function
  // MakeFilePreview
  if (QTInfo_IsRefNumOfResourceFork(theRefNum)) {
    myErr = MakeFilePreview(theRefNum,
theProgressProc);
    goto bail;
  }

  // if the movie has a movie preview, use that as the file preview;
otherwise use
  // a thumbnail of the movie poster frame as the file preview
  if (QTInfo_MovieHasPreview(theMovie))
    myPreviewType = MovieAID;
  else
    myPreviewType = kQTFileTypePicture;

  // construct the 'pnot' atom; fill in the 'pnot' atom header
  myAtomHeader[0] =
EndianU32_NtoB(sizeof(myAtomHeader) +
                                sizeof(myPNOTRecord));
  myAtomHeader[1] =

EndianU32_NtoB(ShowFilePreviewComponentType);

  // fill in the 'pnot' atom data
  GetDateTime(&myModDate);

  myPNOTRecord.modDate = EndianU32_NtoB(myModDate);
myPNOTRecord.version = EndianS16_NtoB(0);
  myPNOTRecord.resType =
EndianU32_NtoB(myPreviewType);
  myPNOTRecord.resID = EndianS16_NtoB(1);

  // write the 'pnot' atom at the end of the data fork

  // get the current logical end-of-file and extend it by the desired amount
  myErr = GetEOF(theRefNum, &myEOF);
  if (myErr != noErr)
    goto bail;

  myErr = SetEOF(theRefNum,
        myEOF + sizeof(myAtomHeader) +
sizeof(myPNOTRecord));
  if (myErr != noErr)
    goto bail;

  // set the file mark
  myErr = SetFPos(theRefNum, fsFromStart, myEOF);
  if (myErr != noErr)
    goto bail;

  // write the atom header into the file
  mySize = sizeof(myAtomHeader);
  myErr = FSWrite(theRefNum, &mySize, myAtomHeader);
  if (myErr != noErr)
    goto bail;

  // write the atom data into the file
  mySize = sizeof(myPNOTRecord);
  myErr = FSWrite(theRefNum, &mySize, &myPNOTRecord);
  if (myErr != noErr)
    goto bail;

  // write the preview data atom at the end of the data fork
  if (myPreviewType == MovieAID) {
    // the 'pnot' atom refers to the existing 'moov' atom
    // so no other preview data atom is required
  }

  if (myPreviewType == kQTFileTypePicture) {
    PicHandle   myPicture = NULL;
    PicHandle   myThumbnail = NULL;

    // get the poster frame picture
    myPicture = GetMoviePosterPict(theMovie);
    if (myPicture != NULL) {
      // create a thumbnail
      myThumbnail = (PicHandle)NewHandleClear(4);
      if (myThumbnail != NULL) {
        myErr = MakeThumbnailFromPicture(myPicture, 0,
                                myThumbnail,
theProgressProc);
        if (myErr == noErr) {
          myAtomHeader[0] =
                    EndianU32_NtoB(sizeof(myAtomHeader)
+

GetHandleSize((Handle)myThumbnail));
          myAtomHeader[1] =
EndianU32_NtoB(myPreviewType);

          // write the atom header into the file
          mySize = sizeof(myAtomHeader);
          myErr = FSWrite(theRefNum, &mySize,
myAtomHeader);
          if (myErr == noErr) {
            // write the atom data into the file
            mySize =
GetHandleSize((Handle)myThumbnail);
            myErr = FSWrite(theRefNum, &mySize,
                                        *myThumbnail);
          }
        }

        KillPicture(myThumbnail);
      }

      KillPicture(myPicture);
    }
  }

bail:
  return(myErr);
}
```

I should point out that QTInfo_MakeFilePreview is not terribly smart about adding file previews to single-fork files. In particular, QTInfo_MakeFilePreview doesn't bother

to check whether the movie file already contains a 'pnot' atom. Instead, it simply appends a new 'pnot' atom and its associated preview data atom to the file. One consequence of this is that each time the user changes any aspect of the movie and saves it, a new thumbnail is appended to the movie file; but that thumbnail might never be used, since StandardGetFilePreview will always find the first 'pnot' atom and the first preview data atom. In the next article, we'll address this issue and see how to replace an existing 'pnot' atom and its associated preview data atom.

Movie Annotations

A QuickTime movie file can include zero or more *movie annotations*, which provide descriptive information about the movie contained in the file. For example, movie annotations can indicate the names of the performers in the movie, the software that was used to create the movie, the names of the movie's writer and director, general information about the movie, and so forth. The header file Movies.h defines over two dozen kinds of movie annotations. For the present, we'll be concerned with only three of them, picked out by these constants:

```
enum {
  kUserDataTextFullName        = FOUR_CHAR_CODE('©nam'),
  kUserDataTextCopyright       = FOUR_CHAR_CODE('©cpy'),
  kUserDataTextInformation     = FOUR_CHAR_CODE('©inf')
}
```

These are the three movie annotations that appear in the movie information dialog box displayed by the ShowMovieInformation function (see Figure 2). What we want to do now is show how to add these three kinds of movie annotations to a QuickTime movie file; or, if a movie file already contains annotations of these sorts, we want to show how to edit those annotations. We'll handle both of these tasks by displaying an Edit Annotation dialog box that contains an editable text field in which the user can add or edit an annotation. For example, if the user selects "Add Information..." in the Test menu of QTInfo but the frontmost movie has no information annotation, we'll display the dialog box shown in **Figure 9**.

Figure 9: *QTInfo's Edit Annotation dialog box*

As you might have guessed from the constants listed above, a movie annotation is stored in a QuickTime movie file as a piece of movie user data. We've already worked a little with the GetMovieUserData, GetUserDataItem, and SetUserDataItem functions for getting and setting a piece of a movie's user data (see "Movie Controller Potpourri" in *MacTech*, February 2000). Because the data for a movie annotation is always text data, here we'll use the GetUserDataText and AddUserDataText functions, which are specialized versions of GetUserDataItem and SetUserDataItem.

When the user selects one of our three menu items for adding or editing a movie annotation, QTInfo executes the QTInfo_EditAnnotation function, passing it a movie identifier and the type of annotation to add or edit. For instance, if the user selects the "Add Information..." item, QTInfo executes this block of code:

```
case IDM_ADD_INFORMATION:
  myIsChanged = QTInfo_EditAnnotation(myMovie,
                                      kUserDataTextInformation);
  if (myIsChanged)
    (**myWindowObject).fIsDirty = true;
  myIsHandled = true;
  break;
```

In the QTInfo_EditAnnotation function, we need to display the Edit Annotation dialog box, put the current movie annotation of the selected kind (if one exists) into the editable text field, allow the user to alter the annotation as desired, and then — if the user clicks the OK button — retrieve the new or edited annotation and attach it to the movie file as a piece of movie

The products you need, with the prices and service you deserve... guaranteed.

Power Tools for Programmers!

CodeWarrior Pro 5

CodeWarrior Professional 5 put everything you need for software development at your fingertips: project management tools, text and resource editors, source and class browsers, compilers, linkers, assemblers, and debugger.Release 5 offers such features as RAD for Java, faster compile times, local and remote application debugging, IDE extensibility options and even tighter C/C++ compliance. Additionally, you can create applications for Windows 95/98/NT and Mac OS 8.x and Mac OS X from either host platform. (available in versions hosted on Mac or Windows).

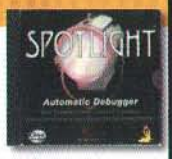
$359

Installer Maker 6.5 10K

Whether you're a developer of high-end, complex applications, simpler utilities, shareware/freeware, an IS manager or an ISP who needs to distribute files quickly and easily, the new InstallerMaker is the complete installation solution for you. Utilizing the power of the new StuffIt Engine, InstallerMaker creates installers faster and smaller than ever, decreasing download time off servers and reducing the number of disks needed to distribute installers. These time and cost savings go straight to your bottom line!

$219

Spotlight



Spotlight is the first Macintosh "Automatic Debugger". It can automatically locate run time errors in your code and display the offending source code line. Unlike similar tools on other platforms Spotlight is easy to use. No source code changes are necessary for application debugging. Spotlight can automatically check for wild pointers, memory leaks, overwrites, underwrites, invalid dereferencing of handles, and even toolbox parameter validity checking -- spotlight knows Macintosh verifying parameters to over 400 toolbox calls.

$189

VOODOO Server

VOODOO Server is a version control system for software developers using Metrowerks CodeWarrior under Mac OS. VOODOO Server and the corresponding VOODOO clients (the included CodeWarrior VCS plug-in and the VOODOO Admin application) are designed to offer reliable and robust version control features while minimizing the administrative overhead that usually accompanies version control. If you're a single programmer or managing a team of developers, version control can make or break your project. Do it right, with VOODOO

$79

Resorcerer 2.2

Resorcerer is the only supported general-purpose resource editor for Macintosh. Relied upon by thousands of Mac developers, Resorcerer features a wealth of powerful yet easy-to-use tools for easier, faster, and safer editing of Macintosh data files and resources. Whether you have to parse a picture, debug a data fork, design and try out Balloon Help, create a scripting dictionary, create anti-aliased icons, design and edit a custom resource with 40,000 fields in it, create C source code to run a dialog, or any of hundreds of other resource-related tasks, Resorcerer's magic will quickly save you time and money.

$256

Future BASIC 3

NEW!

One of the most flexible and powerful development environments on the Macintosh today! Easily create programs with the visual program editor, drop into the BASIC editor to define powerful logic with the worlds easiest programming language, or work directly with the Macintosh toolbox. The only BASIC compiler on the market that gives you 100% access to all of the power of the Macintosh toolbox!

$159

and hundreds more!

Page Charmer 2.0 $139

Scripter 2.0 $179

WebTen $349

FaceSpan 3.0 $179

ToolsPlus Lite $99

WebSpice 1,000,000 $89

WebSpice Animations $89

MkLinux $39

PowerKey Rebound $89

Developer DEPOT®

PO Box 5200 • Westlake Village, CA • 91359-5200 • Voice: 800/MACDEV-1 (800/622-3381)
Outside US/Canada: 805/494-9797 • Fax: 805/494-9798 • E-mail: orders@devdepot.com

www.devdepot.com

Master the Web!

WebSTAR Server Suite 4.2

WebSTAR Server Suite is a complete set of powerful and easy-to-use Internet servers for the Mac OS. Effortlessly serve web pages, host email accounts, publish databases, and share files - all with a single application on one Mac! WebSTAR Server Suite is perfect for Internet or Intranet serving, single or multiple sites, smalll and large businesses - it's power and ease-of-use saves any organization time and money.

$539

CyberGauge 3.0

NEW!

Monitor the bandwidth usage of up to five different machines on your network! Do you need to upgrade your webserver? How hard is your eMail server working? Are you getting all the bandwidth you're paying for? Not only can CyberGauge answer all these questions, new features allow CyberGauge to eMail or page your network device becomes unresponsive r passes a threshold of usage you define – an essential first line of defense for early detection of denial of service attacks and necessity for warning you and tracking quality of ISPs that may have brown outs and shutdowns.

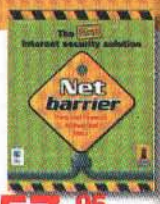
$249

NEW!

Funnel Web is the ultimate web analysis solution for professionals. Specifically designed for profiling web site usage and monitoring customer usage patterns, Funnel Web is ideal for examining server performance and online effectiveness. Funnel Web can analyze log file formats from any server, WebSTAR, WebTen, even Unix or NT hosted ervers. Discover the most popular pages on your site, track server loads & optimize server performance, profile visitors based on organization, domain name, country, browser, etc. Funnel Web does it all!

$299

NetBarrier



NetBarrier offers a Personal Firewall, Antivandal protection, and Internet Filtering. Protect your machine from intrusions by Internet or AppleTalk. Incorrect passwords and individual actions are logged, you are alerted to hostile actions, and intruders are easily isolated. Internet Filtering allows you to be sure that passwords, credit card numbers, and other sensitive information can never be exported from your computer – the content itself is filtered before any transfer! (Rated 4 mice by Macworld Magazine)

$57.95

...and great hardware solutions!

2 USB PCI Card

Add two USB ports to your older Macintosh. Connect up to 127 devices to the Universal Serial Bus (USB) that is Apple's new standard for desktop connectivity. USB mouse devices, keyboards, joysticks, game controllers, printers, scanners — connect them all to your current computer. Installs in minutes!

$32.95

Dr. Bott Moni Switch ADB or USB

NEW!

Do you need 4 monitors and 4 keyboards for your 4 servers? With Dr. Bott Moni-Switch you can connect a single keyboard and monitor to up to 4 machines at once! A simple flick of a switch directs the video input and keyboard commands to the appropriate CPU! Available in USB and ADB models, with 2 or 4 machine support, and bundles with USB PCI cards so you can mix and match USB and ADB machines with the same Moni-Switch! Great for programmers to do back ground compiles, ideal for server rooms overcrowded with monitors and keyboards!

As low as

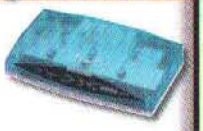
$129.95

Macsense USB Full Size Keyboard

Just plug this keyboard into your Mac and start typing! The UKB-600 keyboard from Macsense is designed to get you typing quickly and easily, without any hassle or compatibility worries. It features two tone translucent design, colored to match your favorite flavor of Macintosh. It offers soft touch with positive tactile feedback and build built in USB port on either side of the keyboard. Includes a 5' USB cable and is 100% Macintosh compatible, simply plug and play, as easy as Macintosh!

$44.95

Macsense Internet Sharing Router



Looking to get your whole office online without shelling out thousands of dollars? If so, the XRouter Internet Sharing Hub offers the perfect solution. This amazing Ethernet-to-Ethernet hub connects an entire network of up to 252 users to the Internet using only one ISP account and one Cable or DSL modem!

$199

user data. Let's consider each of these steps.

Creating the Edit Annotation Dialog Box

Our Edit Annotation dialog box contains four items, as shown in the ResEdit version of our dialog item list ('DITL') resource depicted in **Figure 10**.

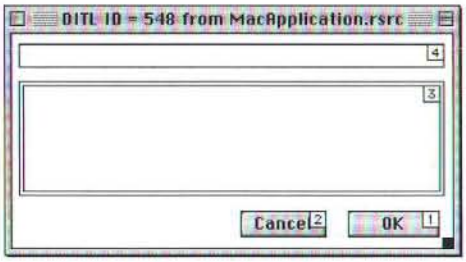


Figure 10: *The dialog item list for the Edit Annotation dialog box*

To be honest, I must admit that I simply "borrowed" this item list from the resource fork of the QuickTime Player application (and I was even too lazy to renumber it). To refer to the items in this dialog box, we'll define these constants:

```
#define kEditTextResourceID          548
#define kEditTextItemOK              1
#define kEditTextItemCancel          2
#define kEditTextItemEditBox         3
#define kEditTextItemEditLabel       4
```

Our resource fork also contains a 'DLOG' resource of the same ID (again "borrowed" from QuickTime Player) that uses this dialog item list. We can open the Edit Annotation dialog box, therefore, by executing this code:

```
myDialog = GetNewDialog(kEditTextResourceID, NULL,
                                        (WindowPtr)-1L);
```

The dialog box is initially invisible, so that we have an opportunity to configure it before displaying it on the screen. For instance, we want to set both the default button (which is outlined in bold and activated when the user types the Return or Enter key) and the cancel button (which is activated when the user types the Escape key or the Command-period key combination). We can do this as follows:

```
SetDialogDefaultItem(myDialog, kEditTextItemOK);
SetDialogCancelItem(myDialog, kEditTextItemCancel);
```

Next, we want to set the static text item (item 4) to indicate which type of movie annotation is being added or edited. I've added a resource of type 'STR#' that contains three strings, one for each of the types of movie annotation that QTInfo can handle. We'll use these constants to access those strings:

```
#define kTextKindsResourceID         2000
#define kTextKindsFullName           1
#define kTextKindsCopyright          2
#define kTextKindsInformation        3
```

We'll simply retrieve one of these strings from that resource, according to the type of annotation that QTInfo_EditAnnotation is asked to handle, as shown in Listing 9.

Listing 9: Setting the label for a movie annotation

QTInfo_EditAnnotation

```
// get a string for the specified annotation type
switch (theType) {
  case kUserDataTextFullName:
    GetIndString(myString, kTextKindsResourceID,
                                kTextKindsFullName);
    break;

  case kUserDataTextCopyright:
    GetIndString(myString, kTextKindsResourceID,
                                kTextKindsCopyright);
    break;

  case kUserDataTextInformation:
    GetIndString(myString, kTextKindsResourceID,
                                kTextKindsInformation);
    break;
}

GetDialogItem(myDialog, kEditTextItemEditLabel,
&myItemKind,
                                &myItemHandle,
&myItemRect);
SetDialogItemText(myItemHandle, myString);
```

As you can see, we call GetDialogItem to get a handle to the static text item and SetDialogItemText to set the string as the text of that item.

Showing the Current Annotation

We also want to call SetDialogItemText to set the current annotation, if one exists, as the text of the editable text item. First, however, we need to find the current annotation of the specified type. As mentioned earlier, we'll use the GetUserDataText function to do this. GetUserDataText reads the movie annotation of a specified type from a user data item list, which we first obtain by calling GetMovieUserData, like this:

```
myUserData = GetMovieUserData(theMovie);
```

GetUserDataText returns the requested information in a handle, which it resizes as necessary to exactly hold the text. So we can retrieve the current movie annotation of the desired type using code like this:

```
myHandle = NewHandleClear(4);
if (myHandle != NULL) {
  myErr = GetUserDataText(myUserData,
                                myHandle, theType, 1,
            GetScriptManagerVariable(smRegionCode));
  // some lines omitted here
}
```

The final parameter passed to GetUserDataText is a *region code*, which specifies a version of a written language of a particular region in the world. It's possible to have several movie annotations of the same type, which differ only in

their region code — that is to say, their language. Here we're using the Script Manager function GetScriptManagerVariable to get the region code associated with the user's current script system.

Once we've called GetUserDataText to get the current annotation of the specified type, we need to copy the text in myHandle into a Pascal string. That's because SetDialogItemText takes a Pascal string as a parameter, not a handle. We can use the function QTInfo_TextHandleToPString, defined in Listing 10, to make this conversion.

Listing 10: Copying text from a handle into a Pascal string

QTInfo_TextHandleToPString

```
void QTInfo_TextHandleToPString (Handle theHandle,
                                 Str255 theString)
{
  short   myCount;

  myCount = GetHandleSize(theHandle);
  if (myCount > 255)
    myCount = 255;

  theString[0] = myCount;
  BlockMoveData(*theHandle, &(theString[1]), myCount);
}
```

So now we are finally ready to insert the existing annotation into the Edit Annotation dialog box. We can do this with these two lines of code:

```
GetDialogItem(myDialog, kEditTextItemEditBox,
                                    &myItemKind,
                  &myItemHandle, &myItemRect);
SetDialogItemText(myItemHandle, myString);
```

The last thing we need to do before displaying the dialog box to the user is set the current selection range of the annotation text. When QuickTime Player displays its Edit Annotation dialog box, it selects all the text in the editable text item. We'll follow this example by calling SelectDialogItemText like this:

```
SelectDialogItemText(myDialog, kEditTextItemEditBox, 0,
                                    myString[0]);
```

At this point, the Edit Annotation dialog box is fully configured. Its static text item has been updated to indicate which type of movie annotation is being edited, and the current annotation of that type has been inserted into the editable text item. We can finish up by actually showing the dialog box to the user:

```
MacShowWindow(GetDialogWindow(myDialog));
```

Retrieving the Edited Annotation

We allow the user to interact with the items in the Edit Annotation dialog box by calling ModalDialog:

```
do {
  ModalDialog(gModalFilterUPP, &myItem);
} while ((myItem != kEditTextItemOK)
       && (myItem != kEditTextItemCancel));
```

As you can see, ModalDialog is called continually until the user clicks the OK or Cancel button (or types a key or key combination that is interpreted as a click on one of those buttons). If the user clicks the Cancel button, we should just exit the QTInfo_EditAnnotation function after disposing of the Edit Annotation dialog box and performing any other necessary clean-up.

```
if (myItem != kEditTextItemOK)
  goto bail;
```

But if the user clicks the OK button, we need to retrieve the text in the editable text item and set it as the movie annotation of the specified type. We can get the edited text like this:

```
GetDialogItem(myDialog, kEditTextItemEditBox,
&myItemKind,
                    &myItemHandle, &myItemRect);
GetDialogItemText(myItemHandle, myString);
```

We want to call AddUserDataText to insert the user's edited annotation into the movie user data list. To do this, we

first need to convert the Pascal string returned by GetDialogItemText into a handle. We can use the QTInfo_PStringToTextHandle function, defined in Listing 11, to handle this conversion.

Listing 11: Copying text from a Pascal string into a handle

QTInfo_PStringToTextHandle

```
void QTInfo_PStringToTextHandle (Str255 theString,
                                 Handle theHandle)
{
  SetHandleSize(theHandle, theString[0]);
  if (GetHandleSize(theHandle) != theString[0])
    return;

  BlockMoveData(&(theString[1]), *theHandle,
theString[0]);
}
```

Now we are ready to call AddUserDataText:

```
myErr = AddUserDataText(myUserData, myHandle, theType,
1,
              GetScriptManagerVariable(smRegionCode));
```

Again, we're calling GetScriptManagerVariable to get the user's current region code, so that the annotation is written into the movie file in a form recognizable to the user. Listing 12 shows the complete function QTInfo_EditAnnotation.

Listing 12: Editing a movie annotation

QTInfo_EditAnnotation

```
Boolean QTInfo_EditAnnotation (Movie theMovie,
                               OSType theType)
{
  DialogPtr   myDialog = NULL;
  short       myItem;
  short       mySavedResFile;
  GrafPtr     mySavedPort;
  Handle        myHandle = NULL;
  short       myItemKind;
  Handle        myItemHandle;
  UserData      myUserData = NULL;
  Rect          myItemRect;
  Str255        myString;
  Boolean     myIsChanged = false;
  OSErr       myErr = noErr;

  // save the current resource file and graphics port
  mySavedResFile = CurResFile();
  GetPort(&mySavedPort);

  // set the application's resource file
  UseResFile(gAppResFile);

  // get the movie user data
  myUserData = GetMovieUserData(theMovie);
  if (myUserData == NULL)
    goto bail;

  // create the dialog box in which the user will add or edit the annotation
  myDialog = GetNewDialog(kEditTextResourceID, NULL,
                          (WindowPtr) -1L);
  if (myDialog == NULL)
    goto bail;

#if TARGET_API_MAC_CARBON
  SetPortDialogPort(myDialog);
#else
  MacSetPort(myDialog);
#endif

  SetDialogDefaultItem(myDialog, kEditTextItemOK);
  SetDialogCancelItem(myDialog, kEditTextItemCancel);

  // get a string for the specified annotation type
  switch (theType) {
    case kUserDataTextFullName:
      GetIndString(myString, kTextKindsResourceID,
                   kTextKindsFullName);
      break;

    case kUserDataTextCopyright:
      GetIndString(myString, kTextKindsResourceID,
                   kTextKindsCopyright);
      break;

    case kUserDataTextInformation:
      GetIndString(myString, kTextKindsResourceID,
                   kTextKindsInformation);
      break;
  }

  GetDialogItem(myDialog, kEditTextItemEditLabel,
                &myItemKind, &myItemHandle,
&myItemRect);
  SetDialogItemText(myItemHandle, myString);

  // set the current annotation of the specified type, if it exists
  myHandle = NewHandleClear(4);
  if (myHandle != NULL) {
    myErr = GetUserDataText(myUserData, myHandle,
theType, 1,

GetScriptManagerVariable(smRegionCode));
    if (myErr == noErr) {
      QTInfo_TextHandleToPString(myHandle, myString);
      GetDialogItem(myDialog, kEditTextItemEditBox,
                    &myItemKind, &myItemHandle,
&myItemRect);
      SetDialogItemText(myItemHandle, myString);
      SelectDialogItemText(myDialog,
kEditTextItemEditBox, 0,
                                          myString[0]);
    }

    DisposeHandle(myHandle);
  }

  MacShowWindow(GetDialogWindow(myDialog));

  // display and handle events in the dialog box until the user clicks OK or
Cancel
  do {
    ModalDialog(gModalFilterUPP, &myItem);
  } while ((myItem != kEditTextItemOK)
         && (myItem != kEditTextItemCancel));

  // handle the selected button
  if (myItem != kEditTextItemOK)
    goto bail;

  // retrieve the edited text
  myHandle = NewHandleClear(4);
  if (myHandle != NULL) {
    GetDialogItem(myDialog, kEditTextItemEditBox,
                  &myItemKind, &myItemHandle,
&myItemRect);
    GetDialogItemText(myItemHandle, myString);
    QTInfo_PStringToTextHandle(myString, myHandle);
    myErr = AddUserDataText(myUserData, myHandle,
theType, 1,

GetScriptManagerVariable(smRegionCode));
    myIsChanged = (myErr == noErr);
    DisposeHandle(myHandle);
  }

bail:
  // restore the previous resource file and graphics port
  MacSetPort(mySavedPort);
```

```
  UseResFile(mySavedResFile);

  if (myDialog != NULL)
    DisposeDialog(myDialog);

  return(myIsChanged);
}
```

Note that QTInfo_EditAnnotation returns a Boolean value that indicates whether the user clicked the OK button and the specified movie annotation was successfully updated. QTInfo uses that value to determine whether it should mark the movie as dirty (and hence in need of saving). It's possible, however, that the user clicked the OK button without having altered the movie annotation in the editable text item. In that case, the movie would be marked as dirty even though its user data has not actually changed. It would be easy to modify QTInfo_EditAnnotation so that it compares the original annotation and the annotation later retrieved from the text box to see whether they differ. This enhancement is left as an exercise for the reader. (It's worth noting, however, that the behavior of QTInfo in this regard is identical to that of QuickTime Player.)

Conclusion

In this article, we've learned how to get and set some of the information that's stored in a QuickTime movie file. We've seen how to work with movie posters and movie previews, and we've seen how to add file previews to both double-fork and single-fork QuickTime movie files. We still have a little bit of work to do on the QTInfo_MakeFilePreview function (which we've deferred until the following article), but already it can write file previews into single-fork movie files.

We've also seen how to add annotations to a movie file and edit a file's existing annotations. Our sample application QTInfo allows the user to edit any of the three kinds of annotations displayed in the movie information dialog box. With just a little bit of work, however, the QTInfo_EditAnnotation function could be modified to support editing *any* type of movie annotation. So what we've got are the essential elements of a general-purpose tool for adding and changing any text-based movie user data.

But, as I've said, we still have some work to do to clean up one or two loose ends in this month's code. Next time we'll see how to find specific atoms in a QuickTime movie file. We'll also discover another kind of atom structure that can be found lurking deep inside QuickTime movie files.

Credits

Thanks are due to **Jim Luther** for pointing me in the right direction in the QTInfo_IsRefNumOfResourceFork function (in the file QTInfo.c) and to **Brian Friedkin** for clarifying the behavior of StandardGetFilePreview under Windows. MT

By Erick J. Tejkowski

REALbasic Sprites

Develop powerful games in minutes

Introduction

Game programming has long been a tradition of highly specialized source code, libraries, and programming trickery, strictly for übergeeks. Until now. Armed with a copy of REALbasic and this article, you will be writing your own games in no time at all.

For high-speed games, it is customary for programmers to use special code to draw at high speeds (often referred to as "blitting" routines). This stems from the fact that the native drawing routines available in the Mac Toolbox are too slow for fast drawing, such as that required by animation. Beginners will be glad to learn that REALbasic has these abilities built in. With only a few lines of code, a programmer can attain high-speed sprite-based animation. Add a few more lines of code and you have a game. REALbasic takes care of all the nasty drawing routines behind the scenes and lets us focus on the elements of the game. This article will demonstrate animation and game creation using REALbasic sprites. By the end of the article, you should be able to complete a small arcade-style game.

Building the Sprites

Begin the game project by opening your favorite resource editor. For our purposes, the free ResEdit from Apple will do just fine. Create a new file and name it "Resources". This is the only name REALbasic understands for resource files added directly to projects in the IDE, so be sure to spell it correctly. In this new resource file, create three resources of type 'cicn' and change the ID# to 1280, 1290, and 1300 respectively by selecting the Resource:Get Resource Info menu. In cicn #1280, create a graphic for the hero of our game. In cicn #1290, draw an adversary. Finally, cicn #1300 contains a picture of a projectile. For this example, we will use an airplane theme. By now, you should have something that looks like **Figure 1**.

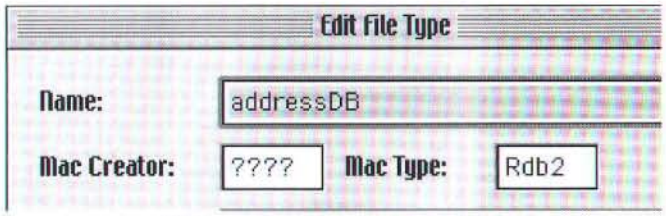


Figure 1. *The Sprite Graphics.*

Now that the sprite graphics have been created, save the Resources file and start REALbasic.

Building the Project

When REALbasic starts, a new project is created automatically. Drag the newly created resource file into the project. Now, open Window1 and drag a StaticText , a PushButton , and a SpriteSurface control onto the window. The StaticText will display the results of the game (i.e. win or lose), the PushButton will start the game, and the SpriteSurface will control all of the animation and game play. **Figure 2** shows the completed interface.

Erick Tejkowski is still looking for a decent version of Moon Patrol™ for the Mac. You can reach him at ejt@norcom2000.com.

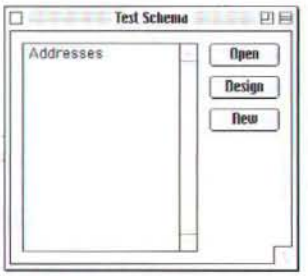


Figure 2. *The Completed Window1 Interface.*

Once the interface has been built, double click Window1 to open the Code Editor. With the Code Editor opened, create the Properties in **Listing 1** by selecting New Property from the Edit menu.

Listing 1. Window1 Properties.

```
BadguySprite(10) as Sprite
GoodGuySprite as Sprite
GunSprite as Sprite
gunvisible as Boolean
NumOfBadGuys as Integer
leftEdge as Integer
topEdge as Integer
```

Sprite properties are pictures that can be controlled by a SpriteSurface. We will need 10 sprites representing the attacking enemies, one representing the hero, and one used for the gunfire that the hero can shoot. The Boolean variable gunvisible will allow us to turn the gunfire on and off. The integer NumOfBadGuys , as one might expect, will keep track of the number of BadguySprites remaining. The final two integer variables (leftEdge and topEdge) will be used to center the SpriteSurface on the screen.

In addition to these properties, create three methods named InitSprites, ClearAllSprites, and RemakeGun by selecting New Method from the Edit menu. The InitSprites method will initialize all of the sprites defined in **Listing 1**. Open the Code Editor to the InitSprites method and enter the code from **Listing 2**.

Listing 2. The InitSprites **Method.**

```
Sub InitSprites()
dim p as picture
dim i as integer

//init the good guy
p=app.resourceFork.getcicn(1280)
GoodGuySprite=SpriteSurface1.NewSprite(p,32,32)
GoodGuySprite.x=300
GoodGuySprite.y=435
GoodGuySprite.group=1

//init the gunfire
p=app.resourceFork.getcicn(1300)
GunSprite=SpriteSurface1.NewSprite(p,32,32)
GunSprite.x=200
GunSprite.y=467
GunSprite.group=2
gunvisible=false

//init the bad guys
p=app.resourceFork.getcicn(1290)
for i=1 to 10
BadguySprite(i)=SpriteSurface1.NewSprite(p,32,32)
BadguySprite(i).x=rnd*600
BadguySprite(i).y=rnd*200-250
BadguySprite(i).group=3
next

NumOfBadGuys=10
End Sub
```

The InitSprites method demonstrates how each of the Sprites is created. First, the icons created earlier are opened into a Picture object. Next, the NewSprite method of SpriteSurface1 is called, passing the Picture, and Width and Height of the Picture as parameters. Once the Sprite has been created, its X and Y positions are initialized. The SpriteSurface will measure 640x480, so the GoodGuySprite will begin life somewhere near the bottom middle of the screen. The gunfire starts off as not being visible (i.e. gunvisible=false), so we assign its X postion somewhere off the screen (GunSprite.x=-200). Later, when we want it visible to the user, we will simply move it to a positive X position. The ten BadguySprites will begin at random X positions and Y positions somewhere between –250 and –50, making them initially invisible. This will change later, however, when they move down the screen towards the GoodyguySprite. The last thing to notice here is the Group property of each of the Sprites. The Group number is used for detecting collisions. Sprites with the same Group number will not collide with each other. Sprites with different Group

numbers produce collisions. Now, when we talk about producing collisions, what this really means is that the Collision Event of the SpriteSurface will be fired. For now, it is sufficient to know that the GunSprite and the BadGuys have different Group numbers so that they may collide. Further, the BadGuySprites have a Group number that is different from the GoodguySprite, allowing them to collide.

Since all good things must come to an end, we need to include some code to destroy everything we have created. Open the Code Editor, navigate to the ClearAllSprites method, and enter the code in **Listing 3**. During the execution of the game, we may need to kill a sprite (e.g. when GunSprite and badguySprite collide). Therefore, we should not assume that all sprites will always be around. This is the reason we check for nil status of a sprite before trying to kill it. If we try to kill a non-existent sprite, an error will result causing the application to crash.

Listing 3. The ClearAllSprites **Method.**

```
Sub ClearAllSprites()
dim i as integer
//kill the GoodguySprite
if GoodGuySprite<>nil then
GoodGuySprite.close
end if
//kill the GunSprite
if GunSprite<>nil then
GunSprite.close
end if
//kill the BadguySprites
for i=1 to 10
if BadguySprite(i)<>nil then
BadguySprite(i).close
end if
next
End Sub
```

One final auxiliary method to code is called RemakeGun. This method will allow us to recreate a GunSprite in the middle of game play. Although we already created one in the InitSprites method, it might be destroyed during game play when it collides with a BadguySprite. Open the Code Editor window, navigate to the RemakeGun method, and enter the code in **Listing 4**.

Listing 4. The RemakeGun **Method.**

```
Sub RemakeGun()
dim p as picture
//reinitialize the gunfire
p=app.resourceFork.geticon(1300)
GunSprite=SpriteSurface1.NewSprite(p,32,32)
GunSprite.x=-200
GunSprite.y=467
GunSprite.group=2
gunvisible=false
End Sub
```

With the window's properties and methods defined, it is time to add functionality to the various controls. To begin, fill the screen with Window1 by adding code into its Open event. This is really only cosmetic in nature. The SpriteSurface will later take over the whole screen anyway. This is what the player will see before the game begins and in between rounds.

```
Sub Open()
me.top=0
me.left=0
me.width=screen(0).width
me.height=screen(0).height
End Sub
```

To get the game started, enter the code in **Listing 5** into the Action Event of PushButton1.

Listing 5. The Action Event **of** PushButton1**.**

```
Sub Action
ClearAllSprites
InitSprites
SpriteSurface1.run
End Sub
```

If any sprites are still around from previous games, they are cleared by the ClearAllSprites method. Next all of the sprites are initialized with InitSprites and the game play begins by calling the Run method of SpriteSurface1.

The SpriteSurface

At the heart of REALbasic sprite animation is the SpriteSurface control. It is a bit of an unusual control in that it has a dual personality. It acts like a Canvas control drawing graphics in a similar fashion, while periodically executing code like a Timer. The FrameSpeed property of the SpriteSurface is the rate at which the timed functions of a SpriteSurface fire. To calculate the FrameSpeed, divide 60 by the desired frames per second and round up to an integer. For example, to achieve a speed 30 frames per second:

```
60 / 30 = 2 (the FrameSpeed)
```

A FrameSpeed of 0 (zero) is the fastest speed at which a SpriteSurface can draw. A setting of 1 or 2 is typical, depending on speed and complexity required. The Run method of SpriteSurface1 was called earlier in the PushButton1 Action event. This causes the SpriteSurface to periodically fire the NextFrame event at the FrameSpeed rate until the SpriteSurface1.close method is called. As the NextFrame event fires, we look at the current conditions of all of the sprites and change them if necessary. For example, it is in the NextFrame event where we check for keys being pressed. If a key is pressed, then some aspect of a sprite can be changed (e.g. position). REALbasic takes care of doing all the redrawing for us. We simply tell it where to draw.

Listing 5 shows the NextFrame event of SpriteSurface1. In it, you will notice some other things going on. First off, a scoreboard is drawn. The scoreboard is simply a string that displays the number of

REAL PEOPLE, REAL SUPPORT.

MacTank provides technical support solutions for Macintosh application developers. We support your end users so you have time to concentrate on marketing and development. Bring your applications to OS X from Windows or NeXT and we will do the rest.

For pricing and information call 877-751-2300, option 2.

Visit us at www.mactank.com

MACTANK
THE TECH SUPPORT ALTERNATIVE

BadguySprites remaining. Behind this string is a small blue rectangle, which serves the purpose of erasing the number each time. The color blue is used, because it will also be the color of the entire background of the SpriteSurface. After redrawing the scoreboard, the code checks the keyboard for keys being pressed. It looks at the left, right, up, and down arrow keys, as well as the space bar. If any of them are depressed, appropriate actions are taken. One piece of code specific to the GunSprite here is ShootingSound.play. ShootingSound is a sound file that has been dragged into the project window. Be sure to make your sounds relatively short, because long sounds can impede smooth animation. Finally the BadGuySprites are moved.

Listing 5. The SpriteSurface1 NextFrame **event.**

```
Sub NextFrame()
dim i as integer
//draw the scoreboard
me.graphics.forecolor=rgb(0,0,100)
me.graphics.fillrect 150+leftEdge,467+topEdge,40,13
me.graphics.forecolor=rgb(255,255,255)
me.graphics.textfont="Geneva"
me.graphics.textsize=10
// ! put the following code all on one line !
me.graphics.drawstring "Bad Guys Remaining: " +
str(NumOfBadGuys), 50+leftEdge,477+topEdge

if me.keyTest(123) then //check left arrow key
//plane moves left
GoodGuySprite.x=GoodGuySprite.x-10
if GoodGuySprite.x<=0 then
GoodGuySprite.x=2
end if
end if
if me.keyTest(124) then //check right arrow key
//plane moves right
GoodGuySprite.x=GoodGuySprite.x+10
if GoodGuySprite.x>600 then
GoodGuySprite.x=598
end if
end if
if me.keyTest(126) then //check up arrow key
//plane moves up
GoodGuySprite.y=GoodGuySprite.y-4
if GoodGuySprite.y<350 then
GoodGuySprite.y=350
end if
end if
if me.keyTest(125) then //check down arrow key
//plane moves down
GoodGuySprite.y=GoodGuySprite.y+4
if GoodGuySprite.y>435 then
GoodGuySprite.y=435
end if
end if

if me.keyTest(49) then //fire=spacebar
//shoot the gun
ShootingSound.play
if gunvisible=false then
//init the gunfire position here
GunSprite.x=GoodGuySprite.x+12
GunSprite.y=GoodGuySprite.y
gunvisible=true
end if
end if

//now check the position of gun only if it's turned on
if gunvisible=true then
GunSprite.y=GunSprite.y-30
//if the gun has reached the top of the screen
//hide it again and turn it off
if GunSprite.y<0 then
GunSprite.x=-200
GunSprite.y=447
gunvisible=false
end if
end if

//advance the badguys
for i=1 to 10
//this is the speed of descent (increase for faster movement)
BadguySprite(i).y=BadguySprite(i).y+5
//if we have reached the bottom of the screen
//go back up to the top at some random x position
if BadguySprite(i).y>447 then
BadguySprite(i).x=rnd*600
BadguySprite(i).y=40
end if
next

End Sub
```

So far, this code will produce a nice animation where the GoodguySprite can navigate and shoot at the approaching BadguySprites.

SpriteSurface Collisions

When a GunSprite hits a BadguySprite, nothing happens yet. Nor will anything occur if the BadguySprite manages to run into the GoodguySprite. This is the function of the SpriteSurface1 Collision Event. **Listing 6** details the code for the SpriteSurface's Collision Event. This is where the Sprite Group numbers from earlier in this article become important. The Collision Event of a SpriteSurface is automatically passed the Group numbers of the sprites that have collided. In the event, we simply check to see which two sprites have collided and then take some action. In this example, we will only look at two types of collisions — both the GunSprite and BadguySprite have collided or the GoodguySprite and BadguySprite have collided. If the GunSprite has collided with a BadguySprite, then we close both of the sprites (in effect "killing" them), play a sound, decrease the number of NumOfBadGuys variable, and call RemakeGun for the next time a user fires the gun. We must also check to see if all of the BadguySprites have been killed. If so, then the game is over and the player has won. Finally, if a BadguySprite has managed to collide with the GoodguySprite then the game is over and the player has lost.

Listing 6. The Collision Event **of** SpriteSurface1.

```
Sub Collision (s1 as Sprite, s2 as Sprite)
//GunSprite and BadguySprite have collided
if s1.group=2 and s2.group=3 then
BadGuyCrashSound.play
s1.close
s2.close
RemakeGun // since we killed the gun sprite, init it again
NumOfBadGuys=NumOfBadGuys-1
if NumOfBadGuys=0 then
s1.close
s2.close
spritesurface1.close
Statictext1.text="You Win!"
CheeringSound.play
end if
end if
```

```
//GoodguySprite and BadguySprite have collided
if s1.group=1 and s2.group=3 then
s1.close
s2.close
spritesurface1.close
Statictext1.text="You Lose!"
GoodGuyCrashSound.play
end if
End Sub
```

To spice things up a little, it is often desirable to draw a background behind the game. The SpriteSurface background is broken up into a grid of 64x64 squares. The PaintTile event takes care of refreshing these background tiles. To fill the background with a solid blue color, enter this code into the PaintTile event of SpriteSurface1.

```
Sub PaintTile (g as Graphics, xpos as Integer, ypos as
Integer)
g.forecolor=rgb(0,0,100)
g.fillrect 0,0,64,64
End Sub
```

Keep in mind that any standard graphics drawing can take place in the PaintTile event.

The final step is to add some code to the Open event of SpriteSurface1. Like the Window1 Open event code, this code is strictly cosmetic. It has been adopted from code by Matt Neuberg in his book *REALbasic: The Definitive Guide*. It centers SpriteSurface1 on the screen.

```
Sub Open()
leftEdge = (Screen(0).width - 640) \ 2
topEdge = (Screen(0).height - 480) \ 2
me.surfaceleft = leftEdge + ((640-me.surfacewidth) \ 2)
me.surfacetop=topEdge
End Sub
```

Having completed the code, it is time to test the finished application. Select Debug:Run menu. Click PushButton1 and play the game. If something goes wrong, recheck your code for accuracy and make sure that you have added all of the necessary auxiliary files (the resource file and the sound files). If all else fails, you can download the finished source code from MacTech's web site.

Conclusions

In this article, we took a brief look at Sprites in REALbasic by constructing a game. The game could be upgraded in a number of ways. Hopefully, this article gives you some basic background about how to make some of these changes yourself. If you would like more information about game development on the Macintosh particularly with regard to REALbasic, be certain to check the Reference section at the end of this article.

References

- REALmac games
  http://www.realmacs.co.uk
  An excellent resource for REALbasic game code and examples.

- REALbasic: The Definitive Guide. Neuberg, Matt. 1999. O'Reilly.
  A standard reference for REALbasic programmers from newbie to advanced. It presents a nice example of an object-oriented sprite game and detailed instructions for REALbasic sprite programming.

- Zune Software
  http://redrival.com/zunezoftware/
  Example games and code.

- REAL RPG
  http://home.earthlink.net/~dathenous/
  REALbasic code site specializing in RPG games.

- REALbasic Arcade Game Project
  http://members.tripod.com/~julianjordan/index.html
  Source Code for a Lunar Lander game.

- REALbasic Monthly
  http://www.nd.edu/~jvanderk/rbm/8-98/index.html
  Introductory REALbasic sprite tutorial.

MT

By Chris Stasny

HTML Rendering with FutureBASIC^3

How to write a simple HTML Browser with FutureBASIC^3

East Meets South

We had been dealing with our Japanese distributors for about six years when their head honcho asked for a face-to-face. It's true that Thelma and I enjoy a life of bliss in our double wide, but I wasn't sure how these foreigners would take to a genuine Mississippi abode. They would have to circumnavigate several junk cars to reach the front door. They would have to sleep with ol' Blue and a half dozen of his flea-bitten companions who hold the existing claim to our guest bed. After a brief emailed discussion of accommodations, we decided to meet in a neutral country: California.

There was a second introduction stacked in the Tarot cards, but this one was binary. I was prodded into action when I discovered that the in-box had been overrun with requests for some method of displaying HTML code. And both of those emails were strongly worded! It was time for... Drum roll... Use deep voice... "*FutureBASIC meets the HTML Renderer.*"

The first step was to locate documentation of the new manager. Apple's only documentation seems to be a terse little reference called HTML_RenderingLib.pdf. A quick search of the hard drive turned up another necessary component. It is an extension called HTMLRenderingLib, which seems to work well in both Systems 8.x and 9. It took me more than an hour to locate the carbon library on a monthly SDK CD that contained information on the constants and toolboxes. Another few minutes were required for converting the code from C to FB^3. These converted toolbox calls are now placed in a file named Tlbx HTML Rendering.Incl and can be found at stazsoftware.com, on the Release 3 CD, or with electronic versions of this publication. The routines are accessed by your program with the following line of code:

```
INCLUDE "Tlbx HTML Rendering.Incl"
```

When Cultures Collide

The whole thing about meeting those foreigners had me on pins and needles. My first faux pas came when they introduced themselves and handed me business cards. They do this by holding the card in both hands and bowing slightly when it is presented. I could tell right off that this was a big deal and I was anxious not to appear uncouth. (These guys were *really* couth.) I hastily extracted a card from my wallet and smoothed it out against my jeans. (This had the added benefit of wiping away some dirt and a few non-descript chicken parts that had adhered to the card.) I scratched out *Bubba's Seafood and Shoe Repair,* then carefully printed my name. I bowed and presented it to that foreign guy. Please note here that things did not work out exactly as I had envisioned. I am loath to admit it, but I believe that a recently consumed six pack of Bud may have been responsible for my falling against him and knocking down the entire Japanese contingent like a row of carefully placed dominoes.

The thought of working with the new HTML Rendering library had me on edge too. I soon discovered that a few simple toolbox calls would do the work for me. To simplify this example program, I decided to use the FutureBASIC II

The only things that Chris Stasny (a.k.a. The STAZ) enjoys more than sitting on the front porch of his trailer are programming his Mac and finding a good place to spit. He has several commercial products under his ample belt which include Classroom Publisher, FutureBASIC^3, and RedNeck Publisher. You can reach the STAZ tech team at tech@stazsoftware.com or visit their web site at www.stazsoftware.com.

runtime (one of the many runtimes available under the **Command** menu). This allowed me to prune window, menu, and event handling so that the example could concentrate on the concepts of HTML rendering. The entire code set (sans remarks and white space) is just over 100 lines. It was written to work with almost any preference setting, but you will need to uncheck **Toolboxes require "CALL"** in the preferences window and make sure that you compile in PPC. (This particular set of routines does not include 68K inline code.) The program operates from a single window and contains a small set of globals.

```
BEGIN GLOBALS
  DIM AS LONG gHRref      // heavily used HR reference
  DIM AS LONG @gPort&     // "@" means don't use register
  DIM gResizeFlag         // bool: window will be resized
  DIM gQuit               // flag says it's time to terminate
END GLOBALS
```

Program brevity may be contributed to the fact that we use only one window. Its pointer is held in gPort&. For non-FBers: You don't have to type class variables in FutureBASIC, though you can if you desire. There is no necessity for setting separate variables for a grafport and a window pointer, as they are (in System 9 and earlier) the same address. Another obvious difference from other languages is that a local function does not have to return a result. Even if it does return something, the caller does not have to accept the result. This makes for a bulletproof environment.

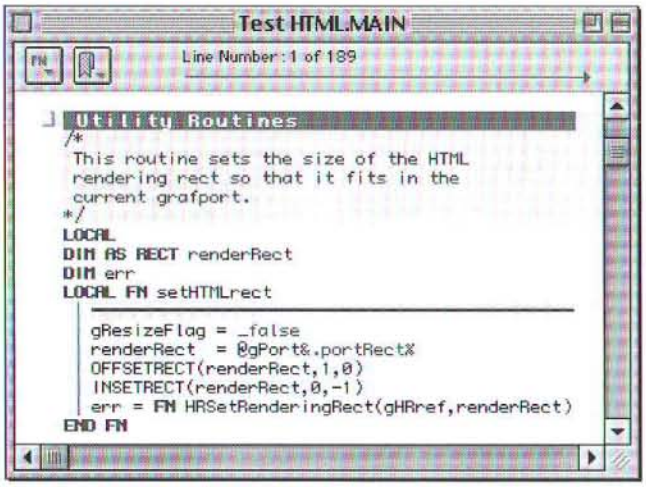
Almost every operation involving the HTML rendering library requires an HTML Rendering (HR) reference number. We start with a single gHRref and use it for the duration. The only remaining globals are two flags that indicate when to resize the rendering area and when to quit the application.

FN setHTMLrect

Three utility functions handle most of the work. The first sets the rectangle of the rendering area based on the size of the grafport. This is called when a window is resized and when the window is initially created.

```
/*
  This routine sets the size of the HTML
  rendering rect so that it fits in the
  current grafport.
*/
LOCAL
DIM AS RECT renderRect
DIM err
LOCAL FN setHTMLrect
  gResizeFlag = _false
  renderRect  = @gPort&.portRect%
  OFFSETRECT(renderRect,1,0)
  INSETRECT(renderRect,0,-1)
  err = FN HRSetRenderingRect(gHRref,renderRect)
END FN
```

Because of automatic colorization and things not used in the formatting of code for this publication, you will see a significant difference in appearance (though not in content) between the printed code and the screen version.



The FB^3 Editor Window.

In FB^3, bookmarks are visible lines instead of obscure selection ranges. Indention and capitalization are automatic. Font, size, style, capitalization, and fore/background colors are user selectable for remarks, bookmarks, keywords, toolbox calls, quoted strings, constants and more. There are several ways of sorting the function menu and you may command-double-click a word to be transported to its definition.

FN showLocalURL

The most important routine calls the toolbox rendering library and handles necessary set up. In FN showLocalURL, we insure that the library is available, create the new HR reference, and open the file specified by name and volume reference number. Note that this particular example works on local files rather than web based files. This lends itself more readily to building HTML based help systems and handling local tests of a web site before uploading. I tested the project by opening a local copy of the STAZ web site and navigating hither and yon. Amazingly, clicking mail links launched my email program. Clicking links to remote locations opened the Netscape browser and took me to the site. If you wish to work from downloaded web pages instead of local files, you will need to review FN HRGoToURL.

```
/*
   Check to see if rendering is available.
   Create a new HR reference.
   Build a file spec to an HTML file.
   Render the file.
*/
LOCAL
DIM spec AS fsspec
DIM err,wTitle$
LOCAL FN showLocalURL(theURL$,vRef%)
  LONG IF FN HRHTMLRenderingLibAvailable

    LONG IF gHRref = 0
      err = FN HRNewReference(gHRref,¬
              _kHRRendererHTML32Type,¬
              gPort&)
      FN setHTMLrect
    END IF

    LONG IF FN FSMAKEFSSPEC(vRef%,0,theURL$,spec) = _noErr
      LONG IF FN HRGoToFile(gHRref,spec,_false,¬
                    _zTrue) = _noErr
        LONG IF FN HRGetTitle(gHRref,wTitle$) = _noErr
          SETWTITLE(gPort&,wTitle$)
        END IF
      END IF
    END IF
  END IF
END FN
```

FN terminate

A final library call performs the simple task of closing down the HR reference. It is called when the application quits.

```
/*
   On exit, dispose of the HR reference.
*/
LOCAL
DIM err
LOCAL FN terminate
  err = FN HRDisposeReference(gHRref)
END FN
```

This Is My Gift to You

Excuse my digression. Only moments ago, we were deeply involved in the tale of an important business meeting. During the last episode, the hero's (that would be me) denim-clad torso was sprawled across a pile of tailored Japanese suits. They declined my offer to help them back to a standing position and proceeded with the next part of eastern culture where we exchanged gifts. They presented a towel thing that had a bunch of that funny looking writing and a small collapsible fan. I wiped sweat from my brow, fanned myself, (to show appreciation) and donned a very large grin with a very small number of teeth. It was then that I realized I was not in possession of a reciprocating gift.

I had to think fast – like the time that Thelma found me in the barn with... (Never mind. I'll save that for another article.) Anyhow, luck was with me because I had just emptied and flattened a perfectly good spit cup prior to the meeting. I turned around, removed it from my shirt pocket, and stretched it back out. I did a perfect military about face in accordance with the pomp and circumstance of the occasion. (Unfortunately, the aforementioned six pack caused me to turn a lot farther than 180 degrees and took a significant number of tiny little baby steps to realign.) I bowed, and presented the head guy with a slightly used (but still perfectly good) spit cup.

Init Routines

The many runtimes of FB^3 are a gift that we programmers may accept without reciprocation. I mentioned earlier that I used the FBII emulation for this project. It allowed me to build and maintain a window with a single line of code. My total set is held in this simple fragment:

```
/*
   Init Everything
*/

_fileMenu = 1

BEGIN ENUM 1
  _openItem
  _quitItem
END ENUM

MENU _fileMenu,0          ,_enable,"File"
MENU _fileMenu,_openItem,_enable,"Open/O"
MENU _fileMenu,_quitItem,_enable,"Quit/Q"

WINDOW 1
GETPORT(gPort&)
```

Dialog Routines

The next task involves event handling. The program accepts the default behaviors for most user interaction, but process a few window message items specific to this type of application. During update events, a region is created and HRdraw is used to redraw the rendered area. When the window is activated or deactivated, HRactivate and HRdeactivate are called into action. When the window is closed, the gQuit Boolean is set and when it is about to be resized, gResizeFlag is set.

```
/*
   Handle window refresh, activate, deactivate,
   grow, and close.
*/
LOCAL
DIM act,ref,err
LOCAL FN doDialog
  act = DIALOG(0)
  ref = DIALOG(act)

  SELECT act

    CASE _wndRefresh       // update
      DIM rgn&
      rgn& = FN NEWRGN
      rectrgn(rgn&,gPort&.portRect%)
      err = FN HRdraw(gHRref,rgn&)
      DISPOSERGN(rgn&)

    CASE _wndActivate      // activate/deactivate
      LONG IF ref > 0
        err = FN HRactivate(gHRref)
      XELSE
        err = FN HRdeactivate(gHRref)
      END IF

    CASE _wndClose         // close
      gQuit = _zTrue

    CASE _preview          // grow
      LONG IF ref = _preWndGrow
        gResizeFlag = _zTrue
      END IF
  END SELECT

END FN
```

FN doEvent

The HR library is intelligent enough to handle its own events. We pass these through a raw event vector which receives events before FB^3 acts upon them.

```
/*
   Handle raw events before FB has a
   opportunity to process them
*/
LOCAL
DIM err
LOCAL FN doEvent
  LONG IF FN HRIsHREvent(EVENT)
    % EVENT,0
  XELSE
    IF gResizeFlag THEN FN setHTMLrect
  END IF
END FN
```

FN doMenu

Menu events are vectored to a single routine. The **Quit** item does nothing more than set a flag. The **Open** item displays a dialog, then vectors to the previously defined FN showLocalURL. Two things are noteworthy.

Noteworthy thing 1: FB^3 places variables in registers until all registers are used. This is an operation that is paramount for PPC speed and is OK until you encounter a parameter that is passed as a pointer to a variable. Registers are not memory locations and cannot themselves be passed as variable addresses. Earlier, we dimensioned @gPort& so that we could use GETPORT(gPort&) because gPort& is a variable that receives information. Similarly, vRef% is dimensioned with the @ symbol because it is a variable that will receive information from the FILES$ routine.

Noteworthy thing 2: Basic users normally use the LEN function to determine the length of a string. In FB^3, you may look at any character in a string by wrapping its offset in brackets. The use of fName$[0] accomplishes the same thing as LEN by extracting the length byte from a Pascal string.

```
/*
   Handle menu events
*/
LOCAL
DIM fName$
DIM @vRef%
LOCAL FN doMenu
  LONG IF MENU(_menuID) = _fileMenu
    SELECT MENU(_itemID)

      CASE _openItem
        fName$ = FILES$(_fOpen,"TEXT",,vRef%)
        LONG IF fName$[0]
          FN showLocalURL(fName$,vRef%)
        END IF

      CASE _quitItem
        gQuit = _zTrue

    END SELECT
  END IF
MENU
END FN
```

Event Loop

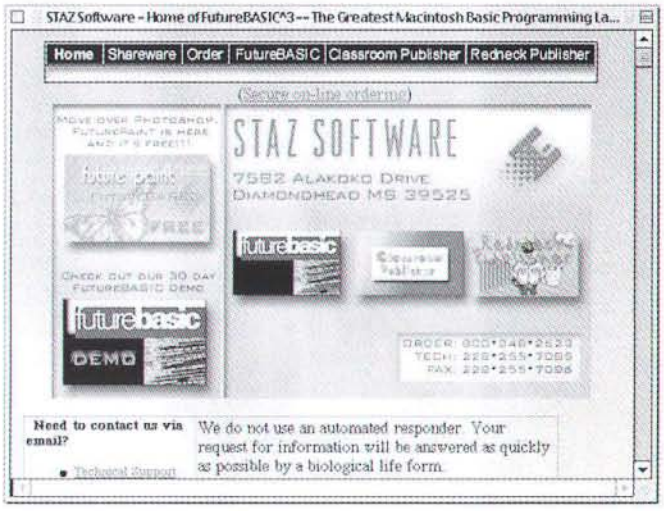
The final code fragment sets up vectors for those events

that interest us, then falls into the event loop.

```
/*
' Event loop
*/

ON DIALOG FN dodialog
ON EVENT  FN doEvent
ON MENU   FN doMenu

DO
  HANDLEEVENTS
UNTIL gQuit

FN terminate
```

A Lean, Mean Rendering Machine

I tested the code by opening the index page in my local copy of a web site. The rendering was clean and generally fast, though background patterns seem to be imaged a bit slower in Apple's library than in Netscape's. Nevertheless, the rendering was exact. No pictures disappeared. No text or formatting was lost. Clicking a graphic or link worked just as it would in Netscape. I had never dreamed that the functionality of a browser could be reduced to a few lines of code just as I never dreamed that meeting a few guys from the other side of the pond would be so unnerving.



Screen Shot of FB^3 Simple Browser.

We have put on the blinders and walked a straight and simple path to the completion of this project. And while it is not your Father's Oldsmobrowser, it is certainly an exciting start into untested waters. Good luck in your quest. MT

Herman Miller Aeron Chair

www.sittingmachine.com

800-883-9697

By Bob Boonstra, Westford, MA

Longest Word Sort

This month's problem puts a twist on a well-known computing problem, the common text sort. Your Challenge is to sort a block of text into ascending or descending order. Yawn.... Except that the comparison function used to do the sort is a little unusual....

The prototype for the code you should write is:

```
void LongestWordSort(
  char *textToSort,          /* the text to be sorted */
  long numChars,             /* the length of the text in bytes */
  Boolean descending,        /* sort in descending order if true */
  Boolean caseSensitive      /* sort is case sensitive if true */
);
```

The textToSort provided to your LongestWordSort routine consists of a sequence of lines of text, each terminated by a return character (0x0D). You are to sort the lines of text in place into either ascending or descending order, based on the value of the descending flag, considering the text as either case sensitive or not case sensitive, based on the caseSensitive flag. The twist is that the primary sort criterion is the length of the words in the line, regardless of the position of the word in the line.

Words are a sequence of alphanumeric characters, separated by anything else (spaces, punctuation, etc.). A line with a longest word of N characters is considered to be "greater" than any line with a longest word of fewer characters. Among lines with longest words of the same length, the sort order is based on the length of the next-longest word, and so on. Finally, among lines with words of exactly the same length, the sort order is based on text order of the individual words, compared in order of length, and then in order of occurrence.

As an example, the following two lines have the same length pattern, with each containing two 5-letter words and one 3-letter word:

```
the quick foxes
early birds win
```

Sorted in ascending order, the second line is smaller, based on a comparison of "early" and "quick".

The text may include non-alphanumeric characters, which should ge ignored for purposes of comparison.

This will be a native PowerPC Challenge, using the CodeWarrior Pro 5 environment. Solutions may be coded in C, C++, or Pascal. This month, we'll also allow solutions that are completely or partially coded in assembly language. The winner will be the solution that correctly sorts a number of blocks of text in the least amount of execution time.

This problem was suggested by Ernst Munter, who earns two Challenge points for the suggestion.

Three Months Ago Winner

The May BigNum Math Challenge required contestants to write a collection of numerical routines for very large integers, numbers with hundreds of digits or more, well beyond what might fit in the memory normally allocated for an integer. The problem required design of an internal representation for these big numbers, and creation of routines to add, subtract, multiply, and divide them. It also required code to raise one such number to the power of another, and code to calculate the square root of such numbers. And it required routines to convert the internal representation back and forth from a decimal representation. Congratulations to **Ernst Munter** (Kanata, Ontario) for taking first place in the BigNum Math Challenge.

Ernst chose to represent his BigNums as a sequence of unsigned 16-bit terms, to ensure that intermediate results fit into a 32-bit long word. The multiplication algorithm is inspired by an algorithm from Knuth, optimized to maximize the use of registers. The exponentiation tests were the most computationally intensive; Ernst's solution scans the bits of the exponent, squaring the intermediate result with each step, and optionally multiplying by the base. For conversion back to decimal, Ernst saves some time by converting to an intermediate representation in base 10000. These and other functions are well commented in the published code.

The second place solution by Mark Day was actually faster than the winning solution in many of the test cases. However, Mark's solution was significantly slower in the exponentiation test case, putting it in second place overall.

The table below lists, for each of the solutions submitted, and the cumulative execution time in milliseconds. It also provides the code size, data size, and programming language used for each entry. As usual, the number in parentheses after the entrant's name is the total number of Challenge points earned in all Challenges prior to this one.

Name	Time (secs)	Code Size	Data Size	Lang
Ernst Munter (607)	1.95	49012	452	C++
Mark Day (20)	4.22	23800	162	C
Tom Saxton (158)	65.1	11700	196	C++
Willeke Rieken (88)	98.63	13044	280	C++
Steve Lobosso	749.06	41608	2813	C++
Jonny Taylor	*	16216	3868	C

Top Contestants

Listed here are the Top Contestants for the Programmer's Challenge, including everyone who has accumulated 10 or more points during the past two years. The numbers below include points awarded over the 24 most recent contests, including points earned by this month's entrants.

Rank	Name	Points	Rank	Name	Points
1.	Munter, Ernst	245	10.	Downs, Andrew	12
2.	Saxton, Tom	146	11.	Jones, Dennis	12
3.	Maurer, Sebastian	78	12.	Day, Mark	10
4.	Boring, Randy	52	13.	Duga, Brady	10
5.	Shearer, Rob	47	14.	Fazekas, Miklos	10
6.	Rieken, Willeke	45	15.	Hewett, Kevin	10
7.	Taylor, Jonathan	26	16.	Murphy, ACC	10
8.	Heithcock, JG	23	17.	Selengut, Jared	10
9.	Brown, Pat	20	18.	Strout, Joe	10

There are three ways to earn points: (1) scoring in the top 5 of any Challenge, (2) being the first person to find a bug in a published winning solution or, (3) being the first person to suggest a Challenge that I use. The points you can win are:

1st place	20 points
2nd place	10 points
3rd place	7 points
4th place	4 points
5th place	2 points
finding bug	2 points
suggesting Challenge	2 points

Here is Ernst's winning BigNum Math solution:

Bignum.cp
Copyright © 2000
Ernst Munter

```
/*
                "BigNum Math"

Version 3.1

Task
----
Implement a set of arithmetic function for multi-digit numbers.
The design of the big number data structure is unspecified.

Data Structure
--------------
BigNum contains a length field (number of decimal digits required to represent the
number) and an opaque (void) pointer to a data structure. I have defined a simple
structure for the internal representation of big numbers:
struct MyNum {
   unsigned short term[]; is the binary representation of the absolute value,
      stored least significant first in unsigned 16-bit chunks.
      The size of term[] is allocated as required by the magnitude of the big number.
   unsigned long numTermsAllocated; records the number of terms allocated.
      a sign bit is hidden as the MSB of numTermsAllocated.
   long lastTerm; records the index of the highest term actually in use.
};

Two special values are defined for BigNum:
  - the numeric value of 0 is represented by setting the bigNumData pointer to 0,
  - an invalid result (e.g. divide by 0) is represented by BigNum.lengthInDigits = -1.

Program Structure
-----------------
The externally available C functions deal directly with most special cases, and then
call a function in a layer which allocates MyNum objects and returns a BigNum.

The allocation layer invokes the appropriate arithmetic method on the MyNum
object to obtain the result which is passed back to the allocation layer, for
conversion
into a BigNum.

Conversion Algorithms
---------------------
Conversions between the decimal digit representation and the binary representation
of MyNum are done in two stages. The intermediate quasi-decimal representation,
uses MyNum formats, but with terms on a base of 10,000 (instead of 65,536). The
routines with the fragment "10k" in their names participate in the conversion
process.

Conversion from decimal to 10k representation is fairly simple: each four decimal
digits make up one 10k term. Conversion from 10k to binary is based on the
expansion
    ...(((x[n]*10k + x[n-1])*10k + x[n-2])*10k .... + x[0],
    evaluated in normal binary arithmetic.

From binary we convert back to the 10k representation in a similar way:
    ...(((y[n]*64K + y[n-1])*64K + y[n-2])*64K .... + y[0],
    but evaluated with base 10k arithmetic.

The splitting of each 10k term into 4 decimal digits is then trivial.

Binary Arithmetic Algorithms
----------------------------
These are commented in the program.
The choice of 16-bit instead of 32-bit terms was primarily made to have a simple
way of dealing with carries and with partial products.

Floating point arithmetic is used in the square root function to obtain a quick good
first estimate from the first few bytes of the input value.

Limitations
-----------
The Power function is limited to providing results within the memory limitations of
a practical machine. The smallest possible base (2) to the power of the largest
unsigned long exponent (about 4 billion) would result in a number of 4 billion bits
(about 1.3 billion digits). Since this alone is beyond the storage capacity (RAM) of
the target machines, I have limited the power function to exponents with at most 32
bits (binary).

Memory for temporary values is allocated using new[]. In the default C++ compiler
mode, an exception will be thrown if memory cannot be allocated, and the program
will just quit.

If you expect to run up to your memory limits, and want to recover from failed
new[]
calls, the program should be modified to either use new with the no_throw option
and test for null, or employ try {} catch() {} blocks.

Version 2 Changes
-----------------
A better initial estimate for square roots is obtained by exploiting the full
power of sqrt(double) to operate on up to 52 integer bits (instead of just 48).

A faster multiplier is implemented which grows at less than the square law with
increasing number size. It is based on the observation that 2 two-digit
numbers can be multiplied with just 3 digit multiplications instead of four.
Used recursively, this turns out to be much faster overall, for large numbers even
though temporary memory must be allocated and initialized frequently, and some
adds and subtracts are needed to combine the terms.

Version 3 Changes
-----------------
The fast multiply function was moved to a separate file "BigMult.cp". The final
recursion step is fanned out to a serious of completely unrolled matrix multipli-
cation routines. This technique alone improved speed again by almost a factor of 3.

Analysis of the disassembled output led to a few small improvements for v.3.1.
Specifically, the use of the struct member "lastTerm" as a loop control resulted
in unnecessary reloads from memory, even though the parameter is declared const.
This inefficiency is avoided by making a local copy of the value, which in turn
leads to the unrolling of the loop by the compiler.
```

```
*/
#define DBG 0

#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <math.h>
#if DBG==0
#define NDEBUG
#endif
#include <assert.h>
#include "bignum.h"
#include "BigMult.h"

typedef unsigned char uchar;
typedef unsigned short ushort;
typedef unsigned long ulong;

typedef uchar* ucharPtr;

// HI and LO are used frequently to separate 32-bit values into 16-bit halves
#define HI(chunk) ((chunk)>>16)
#define LO(chunk) ((chunk)&0xFFFF)

enum {
  kAddMode      = 0,
  kSubMode      = 1,
  kDivideMode       = 0,
  kModuloMode       = 1,
  kPos        = 0,
  kNeg        = 0x80000000, // sign bit constant
  kBase         = 0x10000UL,
  k10k        = 10000,
  kDefaultNumTerms  = 8  // controls minimum size of allocated MyNum
objects
};

// constant for a fairly accurate estimate of decimal digit lengths
const double kDigitsPerBit=log10(2.0);

Binary
static ulong Binary(uchar* digits,int num)
// converts up to 4 decimal digits to binary
// digits are assumed to be binary digits (0-9), NOT digit characters ('0'-'9')
{
  assert((num>=1) && (num<=4));
  ulong v=*digits;
  switch (num)
  {
case 4:v = v * 10 + *++digits;
case 3:v = v * 10 + *++digits;
case 2:v = v * 10 + *++digits;
  }
  return v;
}

// Forward declarations of static functions only as needed
class MyNum;
typedef MyNum* MyNumPtr;
static MyNumPtr MultiplyMyNum(const MyNum & A,const MyNum &
B,int sign);

/*****************************************************
/
/                   Class MyNum
/
/     The class MyNum represents the real data underlying BigNum.bigNumData
/     Most computation and arithmetic work, is done in methods of this class.
/
*****************************************************/
class MyNum {
private:
  ulong numTermsAllocated;      // hides sign bit in MSB of this member
public:
  long lastTerm;            // should be a signed type, initialized to -1
  ushort term[kDefaultNumTerms]; // we store binary terms LSB first

// const functions return computed values without changing MyNum:
  ulong NumTermsAllocated() const {return numTermsAllocated
& ~0x80000000;}
  ulong SignBit() const {return numTermsAllocated &
0x80000000;}
  ulong IsNegative() const {return numTermsAllocated>>31;}
  ulong MSB() const {return term[lastTerm];}
  bool  IsSmall() const {return (0==(lastTerm & ~1));}
      // lastTerm == 0 or 1
  ulong Ulong() const {
      // returns the value of a small MyNum as a 32-bit ulong
    ulong x=term[0];
    if (lastTerm) return x+(term[1] << 16);
    return x;
  }
  ulong IsOdd() const {return term[0] & 1;}
  ulong Log2() const
  {
    ulong lastTermValue=term[lastTerm];
    ulong numBits=lastTerm * 16;
    while (lastTermValue)
    {
      numBits++;
      lastTermValue>>=1;
    }
    return numBits;
  }

  void  SetNumTermsAllocated(ulong numTerms,ulong sign)
    { numTermsAllocated=numTerms | (kNeg & sign);}

  void
ClearMemory(){memset(term,0,sizeof(ushort)*NumTermsAllocated
());}

  void Init(uchar* digits,int numDigits)
// Initializes MyNum from the decimal digits
  {
    int numHighDigits=numDigits % 4;
    uchar* endDigits=digits+numDigits;
    if (numHighDigits)  // Convert first few (<4) digits
    {
        term[0]=Binary(digits,numHighDigits);
      digits+=numHighDigits;
    } else term[0]=0;
    lastTerm=0;
    while (digits<endDigits) // Convert remaining digits in groups of 4
    {
      ulong fourDigits=Binary(digits,4);
      MultiplyAddBy10k(fourDigits);
      digits+=4;
    }
  }

  void MultiplyAddBy10k(ulong chunk)
// Multiplies the current number (base 64K) by 10k and adds chunk to it
  {
    long carry=chunk,i=0,lt=lastTerm;
    for (;i<=lt;i++)
    {
      ulong L=(term[i])*k10k + carry;
      carry=HI(L);
      term[i]=L;
    }
    if (carry)
    {
      lastTerm=i;
      term[i]=carry;
    }
    assert(0== HI(carry));
  }

  void BinaryTo10k(const MyNum & A)
// Converts MyNum from 64K (i.e. binary) base to 10k (i.e. pseudo decimal) base
  {
    const ushort* lastAterm=A.term+A.lastTerm;
    ulong L=*lastAterm,carry=L/k10k;
    term[0]=L%k10k;
    int j=0;
    while (carry)
    {
      L=carry%k10k;
      carry/=k10k;
      term[++j]=L;
    }
```

```
    lastTerm=j;

    while (lastAterm>A.term)
      MultiplyAddMod10k(*--lastAterm);
  }

  void MultiplyAddMod10k(ulong chunk)
// Used in BinaryTo10k. Multiplies MyNum by 64K, using 10k base arithmetic.
  {
    long carry=chunk,i=0,lt=lastTerm;
    for (;i<=lt;i++)
    {
      ulong L=(term[i]<<16) + carry;
      carry=L/k10k;
      term[i]=L%k10k;
    }
    while (carry)
    {
      lt=i;
      term[i++]=carry%k10k;
      carry /= k10k;
    }
    lastTerm=lt;
  }

  long k10ToDigits(uchar* digits) const
  {
// Converts 10k based terms to simple decimal digits.
// Note: terms are stored LSB first, digits are stored MSB first
    ulong D=term[lastTerm];
    int i=
      (D>=1000)?4:(D>=100)?3:(D>=10)?2:1;
    switch (i)
    {
case 4: digits[3]=D%10;D/=10;
case 3: digits[2]=D%10;D/=10;
case 2: digits[1]=D%10;D/=10;
case 1: digits[0]=D;
    }

    int lt=lastTerm;
    for (int j=lt-1;j>=0;j--,i+=4)
    {
      D=term[j];
      digits[i+3]=D%10;D/=10;
      digits[i+2]=D%10;D/=10;
      digits[i+1]=D%10;D/=10;
      digits[i]=D;
    }
    return i;
  }

  int CompAbsolute(const MyNum & A) const
// Compares the absolute values of MyNum and A
// Assumes both numbers are normalized (no leading zeros)
  {
    assert(term[lastTerm]);
    assert(A.term[A.lastTerm]);
    int lt=lastTerm;
    if (lt > A.lastTerm) return 1;
    if (lt < A.lastTerm) return -1;
    do {
      if (term[lt] > A.term[lt]) return 1;
      if (term[lt] < A.term[lt]) return -1;
    } while (--lt >= 0);
//    for (int i=lt;i>=0;i--)
//    {
//      if (term[i] > A.term[i]) return 1;
//      if (term[i] < A.term[i]) return -1;
//    }
    return 0;
  }

  void Sum(const MyNum & A,const MyNum & B)
// MyNum = A + B (absolute values)
// Assumes A has at least as many terms as B
// MyNum has enough terms allocated to handle the result
  {
    assert(A.lastTerm>=B.lastTerm);
    ulong L=A.term[0]+B.term[0],carry=HI(L);
    term[0]=L;
```

OS X Development

Ready to take the plunge?

Okay, everyone into the Aqua! What? Can't swim? No problem. Just climb on our back. We've been to the bottom of Apple's new Mac OS X, far beneath its cool Aqua interface. Porting to Carbon. Diving into Cocoa. Firing up PowerPlant. And developing KEXTs, NKEs, and IOKit drivers. If you're working on an OS X project, we can support you at every level.

A deep pool of talent

We founded our firm on the Mac a decade ago. It's in our DNA. Which is why clients such as 3dfx, Apple, and Lexmark come to us for assistance. We specialize in full-service commercial software development and we thrive on challenging projects. The ones that make your head spin and your staff suck for air.

Contact us soon. We'll help you make a splash in the market —cannonball size.

Critical Path Software

711 SW Alder Street, 4th Floor, Portland, OR 97205 503.222.2922, www.criticalpath.com
Portland • San Diego • Tokyo

Robosoft Technologies

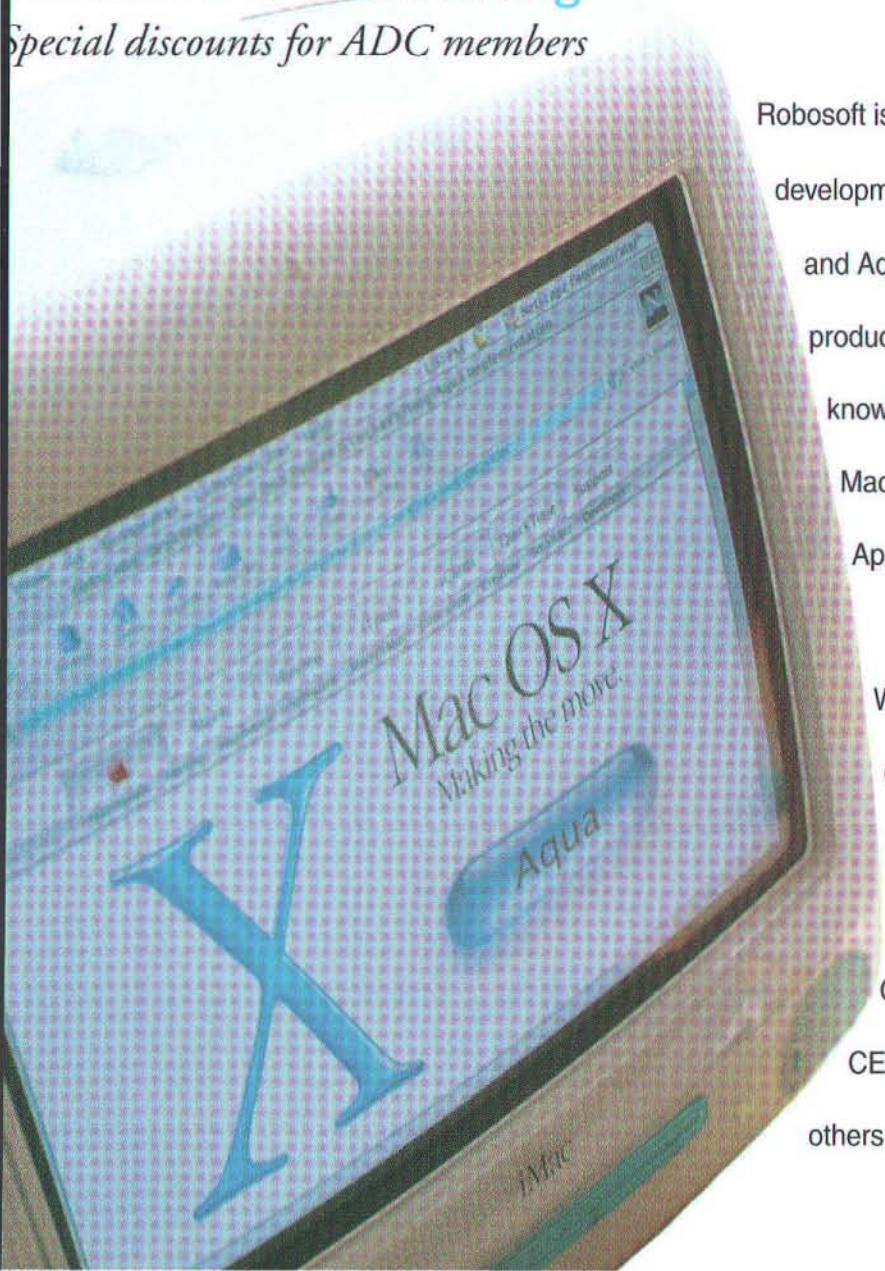
www.robosoftin.com

partners in product development

Indian mind power@ an unbeatable price

Carbon Porting
Aqua Implementation
Windows to Mac Porting

Special discounts for ADC members



Robosoft is an India based specialist in Mac product development, Windows to Mac porting, Carbon porting and Aqua implementation. Our engineers have a strong product development experience with a thorough knowledge of C/C++, Java,Mac OS Toolbox, Macsbug, QuickTime, Carbon API and PowerPlant Application Framework.

We have a proven track record for product development out sourcing and delivering the projects on time. Our clients include a veritable who's who in the industry - Apple, Adobe, Quark, FileWave, Versaware, Veon, NetJumper, CEI, The Anderson Group and eCapital, among others. To know us better, visit our website

>>>>>

info@robosoftin.com

☐ Eat your vegetables.
☐ Exercise every day.
☑ Port to Mac OS X.
☐ Call your mom.

All of these are good for you.
We can make one of them easy.

Since 1989, The Omni Group has worked with the technologies that have been refined into Mac OS X.

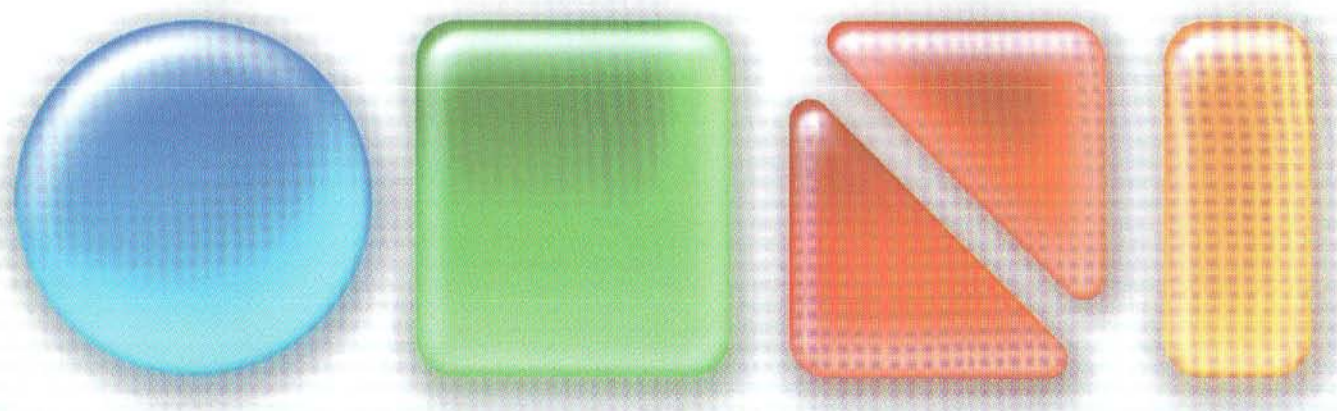
Moving to Mac OS X is a big step for your company, and you need consultants who can help you both plan how best to make the transition and follow whatever path you choose.

That's been our business for years. No matter what kind of product you have, we can get it up under OS X, fast:

- Real games: We ported id's Doom and Quake games to NEXTSTEP and OS X. Quake 2 took us a week.
- Big apps: We ported Adobe's FrameMaker to NEXTSTEP and Sun's Concurrence to OpenStep/Solaris.
- Big libraries: We ported the Oracle 8 client libraries which Apple ships today in OS X Server.
- Serious drivers: We ported 3dfx's Glide and wrote Voodoo2 and Rendition drivers for OS X Server. We've written new mouse drivers for OS X Server and joystick drivers for OpenStep.
- New apps: We wrote OmniWeb, the only native OS X web browser, and OmniPDF, the native Acrobat viewer for OS X.

Mac OS X is what we do. Let us help you do it, too.

The Omni Group



2707 Northeast Blakeley Street
Seattle, Washington 98105-3118
www.omnigroup.com/consulting

sales@omnigroup.com
800.315.OMNI x201
206.523.4152 x201

Get the power and experience you need from

PROSOFT
engineering, inc.

With over thirteen years in providing programming service to the Mac community. Prosoft is committed in delivering the ultimate quality software products and service.

Got OS X?

Our Engineers have extensive knowledge in:

- Macintosh PPC Drivers, Shims and Extensions
- Macintosh Power Plant Applications, Mac OS X applications and drivers
- Carbon application porting for Mac OS X
- Multimedia and QuickTime Applications
- Unix/Linux Console Applications, including Unix Solaris, xBSD Drivers, and Linux Drivers

Other areas of expertise

- Windows 95, 98
- Window CE
- Windows NT, 2000
- Unix / Linux
- Java
- Embedded OS
- RDBMS
- Network Consulting

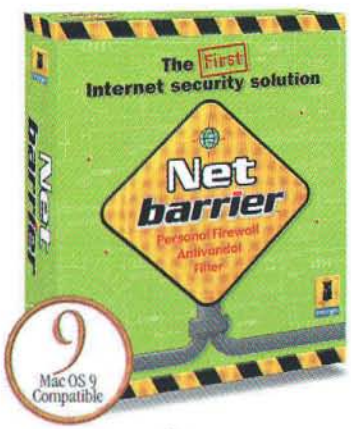
ww.prosofteng.com • tel: (925) 426-6100 • sales@prosofteng.com

```
    int i=1,blt=B.lastTerm;
    for (;i<=blt;i++)
    {
      L=A.term[i]+B.term[i]+carry;
      carry=HI(L);
      term[i]=L;
    }

    int lt=A.lastTerm;
    for (;i<=lt;i++)
    {
      L=A.term[i]+carry;
      carry=HI(L);
      term[i]=L;
    }

    if (carry)
    {
      assert(i < NumTermsAllocated());
      lt=i;
      term[i]=carry;
    }
    lastTerm=lt;
  }

  void Difference(const MyNum & A,const MyNum & B)
// MyNum = A - B (absolute values)
// Assumes A has at least as many terms as B, and A >= B
// MyNum has enough terms allocated to handle A
  {
    assert(A.lastTerm>=B.lastTerm);

    long L=A.term[0]-B.term[0],borrow=HI(L);
    term[0]=L;
    int i=1,blt=B.lastTerm;
    for (;i<=blt;i++)
    {
      L=A.term[i]-B.term[i]+borrow;
      borrow=HI(L);
      term[i]=L;
    }

    int lt=A.lastTerm;
    for (;i<=lt;i++)
    {
      L=A.term[i]+borrow;
      borrow=HI(L);
      term[i]=L;
    }
    // remove leading zeros
    while ((lt>0) && (term[lt]==0)) lt--;
    lastTerm=lt;
  }

  void SmallProduct(const MyNum & B,const MyNum & A,ulong
hiTerm)
// MyNum = A * B (absolute values)
// MyNum has enough terms allocated to handle A*B
  {
    assert(NumTermsAllocated()>=1+hiTerm);
    memset(term,0,(1+hiTerm)*sizeof(ushort));
    int alt=A.lastTerm,blt=B.lastTerm;
    for (int i=0;i<=alt;i++)
    {
      ulong a=A.term[i];
      for (int j=0;j<=blt;j++)
      {
        int k=i+j;
        ulong x=term[k]+a*B.term[j];
        term[k]=x;
        ulong carry=HI(x);
        while (carry)
        {
          assert(k+1<NumTermsAllocated());
          x=term[++k]+carry;
          term[k]=x;
          carry=HI(x);
        }
      }
    }
    // remove leading zeros
    while (hiTerm && (0==term[hiTerm])) hiTerm--;
    lastTerm=hiTerm;
  }

  void Product(const MyNum & A,const MyNum & B);
// MyNum = A * B (absolute values)

  void SmallQuotient(MyNum & R,const ulong divisor)
// Divides R by the small divisor, a single ulong.
// MyNum = R / divisor (absolute values).
// MyNum will be the quotient. R will contain the remainder.
  {
    int ir=R.lastTerm;
    ulong upperDigit=0;
    for (int iq=ir;iq>=0;iq--)
    {
      upperDigit<<=16;
      ulong x=R.term[iq] + upperDigit;
      if (upperDigit)
      {
        R.term[iq+1]=0;
        R.lastTerm=iq;
      }
      if (x < divisor)
      {
        term[iq]=0;
        upperDigit=x;
      } else
      {
        ulong y=x/divisor;
        term[iq]=y;
        if (lastTerm<iq)
          lastTerm=iq;
        R.term[iq]=upperDigit=x - y*divisor;
      }
    }
  }

  void Quotient(MyNum & U,const MyNum & V);

  void DeNormalize(int shift)
// Reverse shift operation for the remainder after (big)Quotient division
  {
    int i=0,lt=lastTerm;
    for (;i<lt;i++)
    {
      ulong v=((term[i+1]<<16) + term[i]) >> shift;
      term[i]=v;
    }
    term[i] >>= shift;
  }

  void Power(const MyNum & A,const ulong e)
// MyNum = A ** e, all values must be positive
// The power is evaluated by scanning the exponent, MSB first.
// For each except the first bit, the current value of MyNum is squared,
// if the bit is a 1, the current value of MyNum is then multiplied by A.
// Multiplication requires a distinct product destination, so a temporary MyNum T
// is allocated. Each multiplication uses either MyNum or T as the source, and
// the other as the destination.
  {
    assert(numTermsAllocated>A.lastTerm);
    assert(e != 0);

    // copy A to this
    lastTerm=A.lastTerm;
    memcpy(term,A.term,(1+lastTerm)*sizeof(ushort));

    if (e==1) return;

    // find MSB of the exponent
    ulong mask=0x80000000;
    while ((e & mask) == 0) mask >>= 1;

    mask >>= 1;

    // make a second MyNum to hold intermediate products
    int allocatedSize = (numTermsAllocated-kDefaultNumTerms)
*
        sizeof(ushort) + sizeof(MyNum);
    MyNumPtr T=MyNumPtr(new uchar[allocatedSize]);
    T->numTermsAllocated=numTermsAllocated;

    // we'll multiply by ping-ponging between MyNum and *T
    // but the result should end up in MyNum
```

```
    MyNumPtr temp[2] = {this,T};

    int k=0;
    while (mask) // for each additional bit in e
    {
      // always square
      temp[1-k]->Product(*temp[k],*temp[k]);        k=1-k;
      if (e & mask)
      {
        // multiply by A if bit is set
        temp[1-k]->Product(*temp[k],A);     k=1-k;
      }
      mask >>= 1;
    }

    if (k==1)  // result ended up in *T: must copy to *this
    {
      lastTerm=T->lastTerm;
      memcpy(term,T->term,(1+lastTerm)*sizeof(ushort));
    }
    delete [] T;
  }

  void SquareRootEstimate(const MyNum & A)
// Returns the square root of the three or four most significant terms of MyNum
  {
    assert(A.lastTerm>=2);
    int lt=lastTerm=A.lastTerm/2;
    double top3Digits = A.Ulong()*65536.0 +
A.term[A.lastTerm-2];

    if (A.lastTerm==2)  // just use the three digits
    {
      ulong root=sqrt(top3Digits);
      // the top 3 digits of A are converted to a 24 bit root
      // the result is loaded into the top 2 digits of MyNum

      if (A.lastTerm & 1) // even number of digits in A
      {
        term[lt] = root >> 8;
        term[lt-1]= root << 8;
      } else
      {
        term[lt] = HI(root);
        term[lt-1]= root;
      }
    } else // use top 51 or 52 bits to get the most from sqrt(double)
    {   // resulting in a 26-bit estimate
      ulong lastTermValue=A.MSB();
      ulong shift=0;
      while (lastTermValue>0xF)  // get minimum even size of MSB word
      {                  // keep at least 4 bits
        shift+=2;
        lastTermValue>>=2;
      }
      ulong factor = 1<<(16-shift);
      double top52Bits=
        top3Digits * factor + (A.term[A.lastTerm-
3]>>shift);

      ulong root=sqrt(top52Bits);
      // the top 4 digits of A (minus shifted bits) are converted to a 26 bit root
      // the result is loaded into the top 2 or 3 digits of MyNum
      shift /= 2;
      if (A.lastTerm & 1) // even number of digits in A
      {
        term[lt] = root >> (16-shift);
        term[lt-1]= root << shift;
      } else
      {
        term[lt]   = root >> (24-shift);
        term[lt-1]= root >> (8-shift);
        term[lt-2]= root << (shift+8);
      }
    }
  }

  ulong Average(MyNum & A)
// MyNum becomes the average of MyNum and A.
// Returns true if the original MyNum and the new average are different.
  {
    assert(lastTerm==A.lastTerm);

    ulong carry=0;
    ulong compare=0;  // detect and track changes in compare
    for (int i=lastTerm;i>=0;i--)
    {
      ulong oldTerm=term[i];
      ulong x = (carry + oldTerm + A.term[i]);
      carry = (1 & x) << 16;
      x >>= 1;
      term[i]=x;
      compare |= oldTerm ^ x;
    }
    return compare & 0xFFFF;
  }

  long EstimateLength() const
// The length in decimal digits is estimated from the number of bits used.
// The result should always be within 1 digit of the true count.
  {
    double numDigits=Log2()*kDigitsPerBit;
    return 1+long(numDigits);
  }

  void Normalize()
// Removes leading zeros
  {
    int k=lastTerm;
    while ((k>0) && (term[k]==0)) k--;
    lastTerm=k;
  }

  BigNum Convert2BigNum();
// Returns the BigNum representation of MyNum
};

// The following MyNum functions are not inlined (reduced size, no loss of speed):

BigNum MyNum::Convert2BigNum()
// Returns the BigNum representation of MyNum
  {
    Normalize();
    BigNum result;
    result.bigNumData=this;
    result.lengthInDigits=EstimateLength();
    return result;
  }

void MyNum::Product(const MyNum & A,const MyNum & B)
// MyNum = A * B (absolute values)
// MyNum has enough terms allocated to handle A*B
  {
    ulong alt=A.lastTerm,blt=B.lastTerm,hiTerm=alt+blt;
    assert(NumTermsAllocated()>=2+hiTerm);
    assert(alt >= blt);
    ulong aSize=alt+1,bSize=blt+1;
    if (bSize >= kMultiplyThreshold)
    {
// Big numbers use fast multiply which requires N to be a small
// integer (<=15) times a power of two.  RoundUp ensures this.
      ulong N=A.lastTerm+1;
      N=RoundUp(N);
// Allocate memory for:
//    factor U N terms  (copy and pad A to U)
//    factor V N terms  (copy and pad B to V)
//    product P   2N terms (compute P = U*V and copy result back to MyNum)
//    scratch     4N terms needed for intermediate products
      ushort* U=new ushort[8*N];
      ushort* V=U + N;
      ushort* P=V + N;
      memcpy(U,A.term,aSize*sizeof(ushort));
      memset(U+aSize,0,(N-aSize)*sizeof(ushort));
      memcpy(V,B.term,bSize*sizeof(ushort));
      memset(V+bSize,0,(N-bSize)*sizeof(ushort));

      Multiply(U,V,P,N,bSize);

      // remove leading zeros from P
      N += N-1;
      while (N && (0==P[N])) N--;

      assert(N<NumTermsAllocated());
      memcpy(term,P,(1+N)*sizeof(ushort));
      lastTerm=N;
      delete [] U;
```

You think the Internet is safe. Think again...

NetBarrier. The first Internet security solution for Macintosh.



All Macs connected to the Internet (dialup, DSL, cable-modem) are exposed to hackers. Whether you are a home user or a professional user, your data interests them. That's why you need a security solution that only NetBarrier can provide.

Personal Firewall

NetBarrier protects and monitors all incoming and outgoing data. A customized mode allows you to create your own defense rules, thereby offering the most secure level of protection.

Antivandal

NetBarrier blocks all attempts to break into your Mac, detects wrong passwords and logs vandal attacks for complete protection. Moreover, it has an alarm to inform you of every intrusion attempt.

Internet filter

NetBarrier analyzes data as it leaves your computer and prevents unauthorized exporting of private information such as credit card numbers, passwords, sensitive data and more...

Available @ Developer DEPOT
<http://www.devdepot.com>
Toll Free: 877-DEPOT-NOW
Outside U.S./Canada: 805/494-9797

www.intego.com

intego

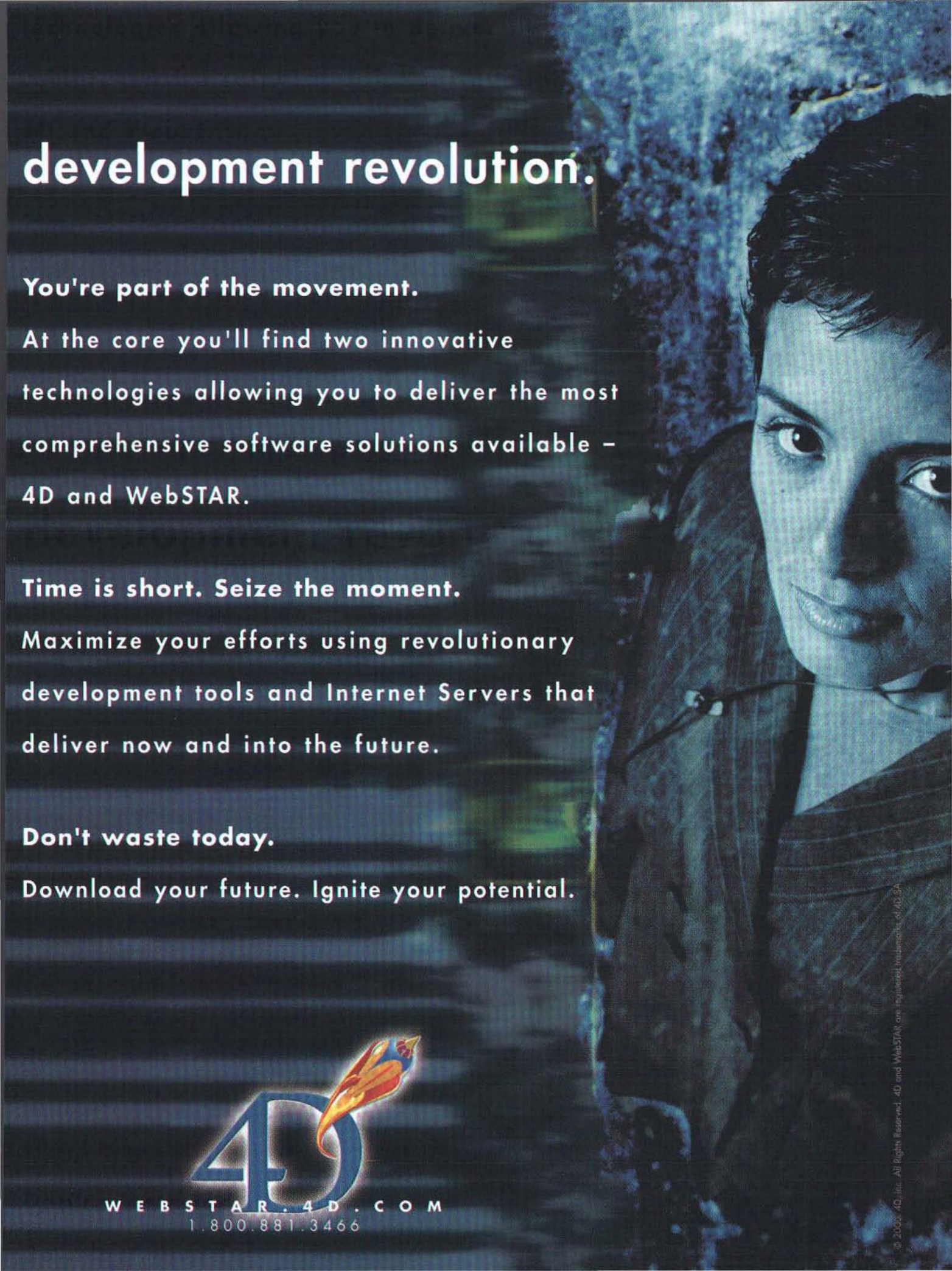
MacNN ★★★★

Macworld

Macworld 4/5

APPLE WIZARDS ★★★★★

```
    } else
// The smallest factor is not so large, we use a simple multiplication loop

      SmallProduct(A,B,1+hiTerm);
    }

void MyNum::Quotient(MyNum & U,const MyNum & V)
// Divides U by the MyNum divisor V.
// MyNum will contain the quotient, U the remainder.
// The implementation closely follows Donald Knuth's algorithm 'D' (Vol.2, page 272)
    {
      // U and V already normalized by caller, i.e. V.MSB >= kBase/2
      int n=1+V.lastTerm, // size of the divisor
        m=U.lastTerm-n, // msb index of the quotient
        k=U.lastTerm, // msb dividend index
        j=m;        // qotient index

      assert(n>=2);   // for smaller divisors, should use SmallQuotient()

      ulong divisorMSB = V.term[V.lastTerm];
      for (;j>=0;j--,k--)
      {
        ulong x= ulong(U.term[k])*kBase + U.term[k-1];

        ulong q= x / divisorMSB;
        ulong r= x - q * divisorMSB;
        if ((q==kBase) || (q*V.term[n-2] > (kBase*r +
U.term[k-2])))
        {
          q--; r+=divisorMSB;

        }

        ulong p,carry=0,borrow=0;
        int i=0;
        for (;i<n;i++)
        {
          p = carry + q * V.term[i];
          x=U.term[j+i] - LO(p) + borrow;
          borrow=HI(long(x));
          carry=HI(p);
          U.term[j+i]=x;
        }

        x=U.term[j+i] - LO(carry) + borrow;
        borrow=HI(long(x));
        U.term[j+i]=x;

        term[j]=q;
        if (lastTerm<j) lastTerm=j;
        if (borrow)
        {
          term[j]--;
          carry=0;
          for (int i=0;i<n;i++)
          {
            x=carry + U.term[j+i] + V.term[i];
            U.term[j+i] = x;
            carry=HI(x);
          }
          U.term[j+i] = 0;
        }
      }
// Denormalization to be done by the caller only if the remainder is needed.
    }
/*****************************************************
/
/              Utility functions returning a MyNumPtr
/
*****************************************************/

static MyNumPtr New(int minNumTerms,int sign)
// returns a fresh MyNum, with at least minNumTerms terms
{
  int numTermsAllocated=
    (minNumTerms+kDefaultNumTerms-1) & ~(kDefaultNumTerms-
1);
  assert(numTermsAllocated>=minNumTerms);
  int allocatedSize = (numTermsAllocated-kDefaultNumTerms) *
        sizeof(ushort) + sizeof(MyNum);
  MyNumPtr B=MyNumPtr(new uchar[allocatedSize]);
  B->SetNumTermsAllocated(numTermsAllocated,sign);
  B->lastTerm=-1;
  return B;
}

static MyNumPtr NewCopy(const MyNum & A,int sign)
// returns a fresh exact copy of A, (the sign may differ from the sign of A)
{
  MyNumPtr B=New(A.NumTermsAllocated(),sign);
  B->lastTerm=A.lastTerm;
  memcpy(B-
>term,A.term,A.NumTermsAllocated()*sizeof(ushort));
  return B;
}

static MyNumPtr NewCopyShifted(const MyNum & A,
      int sign,int shift,int extraDigit)
// returns a fresh shifted copy of A, possibly expanded by 1 term
{
  int requestedSize=1+extraDigit+A.lastTerm;// allow for extra
"digit"
  MyNumPtr B=New(requestedSize,sign);
  B->lastTerm=A.lastTerm+extraDigit;
  if (shift==0)   // straight copy, set extra digit to 0
  {
    memcpy(B->term,A.term,(1+A.lastTerm)*sizeof(ushort));
    if (extraDigit) B->term[B->lastTerm]=0;
  } else
  {           // copy shifted.
    const ushort* at=A.term;
    const ushort* lastAterm=at+A.lastTerm;
    ushort* bt=B->term;
    ulong x=(ulong(*at) << shift);
    *bt=x;
    ulong carry=HI(x);
    while (at<lastAterm)
    {
      x=carry + (ulong(*++at) << shift);
      *++bt=x;
      carry=HI(x);
    }
    if (extraDigit) *++bt=carry;
  }
  return B;
}

static MyNumPtr BinaryTo10k(BigNum bigNumA)
// converts a BigNum to a base-10k MyNum
{
  const MyNumPtr A=MyNumPtr(bigNumA.bigNumData);
  MyNumPtr B=New((bigNumA.lengthInDigits+3)/4,A->SignBit());
// allocate 1 ushort for every 4 digits
  B->BinaryTo10k(*A);
  return B;
}

/*****************************************************
/
/          Layer of the arithmetic functions returning a MyNumPtr.
/
/  Functions determine the size of the result and allocate a new MyNum object
/  accordingly.
/
/  Then they call a method of the allocated object for most of the actual arithmetic.
/
*****************************************************/

static MyNumPtr AddMyNum(const MyNum & A,
      const MyNum & B,int sign)
// returns C = A + B
{
  ulong numTermsC=2+A.lastTerm;
  MyNumPtr C=New(numTermsC,sign);
  C->Sum(A,B);
  return C;
}

static MyNumPtr SubMyNum(const MyNum & A,
      const MyNum & B,int sign)
// returns C = A - B
{
  ulong numTermsC=1+(A.lastTerm);
  MyNumPtr C=New(numTermsC,sign);
  C->Difference(A,B);
```


development revolution.
You're part of the movement.
At the core you'll find two innovative technologies allowing you to deliver the most comprehensive software solutions available – 4D and WebSTAR.
Time is short. Seize the moment.
Maximize your efforts using revolutionary development tools and Internet Servers that deliver now and into the future.
Don't waste today.
Download your future. Ignite your potential.
4D
WEBSTAR.4D.COM
1.800.881.3466
© 2000 4D Inc. All Rights Reserved. 4D and WebSTAR are registered trademarks of 4D SA

```
  return C;
}

static MyNumPtr MultiplyMyNum(const MyNum & A,
      const MyNum & B,int sign)
// returns C = A * B,
{
  ulong numTermsC=2+A.lastTerm+B.lastTerm;
  MyNumPtr C=New(numTermsC,sign);
  if (A.lastTerm >= B.lastTerm)
    C->Product(A,B);
  else
    C->Product(B,A);
  return C;
}

static int ShiftNeeded(const MyNum & D)
// returns bit shift needed to ensure MSB of a divisor is >= kBase/2
{
  ulong msb=D.MSB(),shift=0;
  while (msb < (kBase/2)) {msb <<= 1;shift++;}
  return shift;
}

static MyNumPtr DivideMyNum(const MyNum & P,
      const MyNum & D,int sign)
// returns the result of P/D, either
// the quotient Q = P/D,
// or the remainder R = P%D, according to the mode parameter
{
  assert(P.lastTerm >= D.lastTerm);
  long numTermsQ=1+P.lastTerm-D.lastTerm;
  MyNumPtr Q=New(numTermsQ,sign),R;

  if (D.lastTerm==0)  // single "digit" divider
  {
    R=NewCopy(P,sign);
    Q->SmallQuotient(*R,D.term[0]);
  } else          // general case
  {
    int shift=ShiftNeeded(D);
    // Create shifted copies of dividend and divisor
    R=NewCopyShifted(P,sign,shift,1);
    MyNumPtr V=NewCopyShifted(D,sign,shift,0);
    Q->Quotient(*R,*V);
    delete [] V;
  }
  delete [] R;
  return Q;
}

static MyNumPtr ModMyNum(const MyNum & P,const MyNum & D,int
sign)
// returns the result of P/D, either
// the quotient Q = P/D,
// or the remainder R = P%D, according to the mode parameter
{
  assert(P.lastTerm >= D.lastTerm);
  long numTermsQ=1+P.lastTerm-D.lastTerm;
  MyNumPtr Q=New(numTermsQ,sign),R;

  if (D.lastTerm==0)  // single "digit" divider
  {
    R=NewCopy(P,sign);
    Q->SmallQuotient(*R,D.term[0]);
  } else          // general case
  {
    int shift=ShiftNeeded(D);
    // Create shifted copies of dividend and divisor
    R=NewCopyShifted(P,sign,shift,1);
    MyNumPtr V=NewCopyShifted(D,sign,shift,0);
    Q->Quotient(*R,*V);

    delete [] V;
    R->DeNormalize(shift);// undo bit shift
  }
  delete [] Q;
  return R;
}


static MyNumPtr PowerMyNum(const MyNum & A,
      const ulong exponent,int sign)
// returns C = A ** B
{
  ulong numTermsC=2+A.Log2()*exponent/16;
  MyNumPtr C=New(numTermsC,sign);
  C->Power(A,exponent);
  return C;
}

static MyNumPtr SquareRootMyNum(const MyNum & A)
// returns C = sqrt(A)
//The algorithm starts with an estimate and refines the estimate successively
// according to the equation
//
//    C = (C + A/C) / 2
//
// until C no longer changes.
{
  ulong numTermsC=(2+A.lastTerm)/2;
  MyNumPtr C=New(numTermsC,kPos);
  C->ClearMemory();
  C->SquareRootEstimate(A);
  if (A.lastTerm<=2)
    return C;
  int n=0;
  for (;n<=A.lastTerm;n++)
  {
    MyNumPtr D=DivideMyNum(A,*C,kPos);
    D->Normalize();
    if (0==C->Average(*D))
    {
      delete [] D;
      break;
    }
    delete [] D;
  }
  return C;
}

static MyNumPtr AddSubtract(const MyNum & A,
      const MyNum & B,int selection)
//Addition and subtraction cases are classified according to the signs
// and relative sizes of the operands. The resulting 24 possible cases
// can then be treated as add/subtracts of positive numbers only.
{
  switch (selection) {
case 0:case 3:
case 8:case 11:return AddMyNum(B,A,kPos);
case 1:case 2: return SubMyNum(B,A,kNeg);
case 5:case 6: return AddMyNum(B,A,kNeg);
case 4:case 7: return SubMyNum(B,A,kPos);
case 9:case 10:case 12:case 15: return 0;
    // null-pointer represents the value 0.
case 13:case 14:
case 21:case 22:return AddMyNum(A,B,kNeg);
case 16:case 19:return AddMyNum(A,B,kPos);
case 17:case 18:return SubMyNum(A,B,kPos);
case 20:case 23:return SubMyNum(A,B,kNeg);
  };
  return 0;
}

/*************************************************************
/
/       Utility functions returning a BigNum for special cases
/
/*************************************************************/

static BigNum Zero(int length)
// returns the BigNum version of 0 (length = 1)
// or the error code (length = -1)
{
  BigNum result;
  result.lengthInDigits=length;
  result.bigNumData=0;
  return result;
}

static BigNum Copy(BigNum bigNumA,ulong sign)
// returns a fresh copy of bigNumA, sign may change
{
  BigNum result;
  MyNumPtr A=MyNumPtr(bigNumA.bigNumData);
  result.lengthInDigits=bigNumA.lengthInDigits;
  result.bigNumData=NewCopy(*A,sign);
```

Yellow Dog Linux Champion Server™
by Terra Soft Solutions, Inc.

Dedicated to Apple Computer, Terra Soft Solutions maintains the largest and most experienced PowerPC Linux development staff in the world. Enjoy this mature, **proven**, and professional Linux distribution.

In addition to the expected Linux applications and utilities Yellow Dog now includes **Mac-On-Linux** which runs the Mac OS inside of Linux at near native speeds, and **"yup!"**, our famous automated system update utility.

Available on 3 CDs or **pre-installed** on hard drives.
Includes bootable Install & Rescue CDs and all source RPMs.
60 days installation support available.

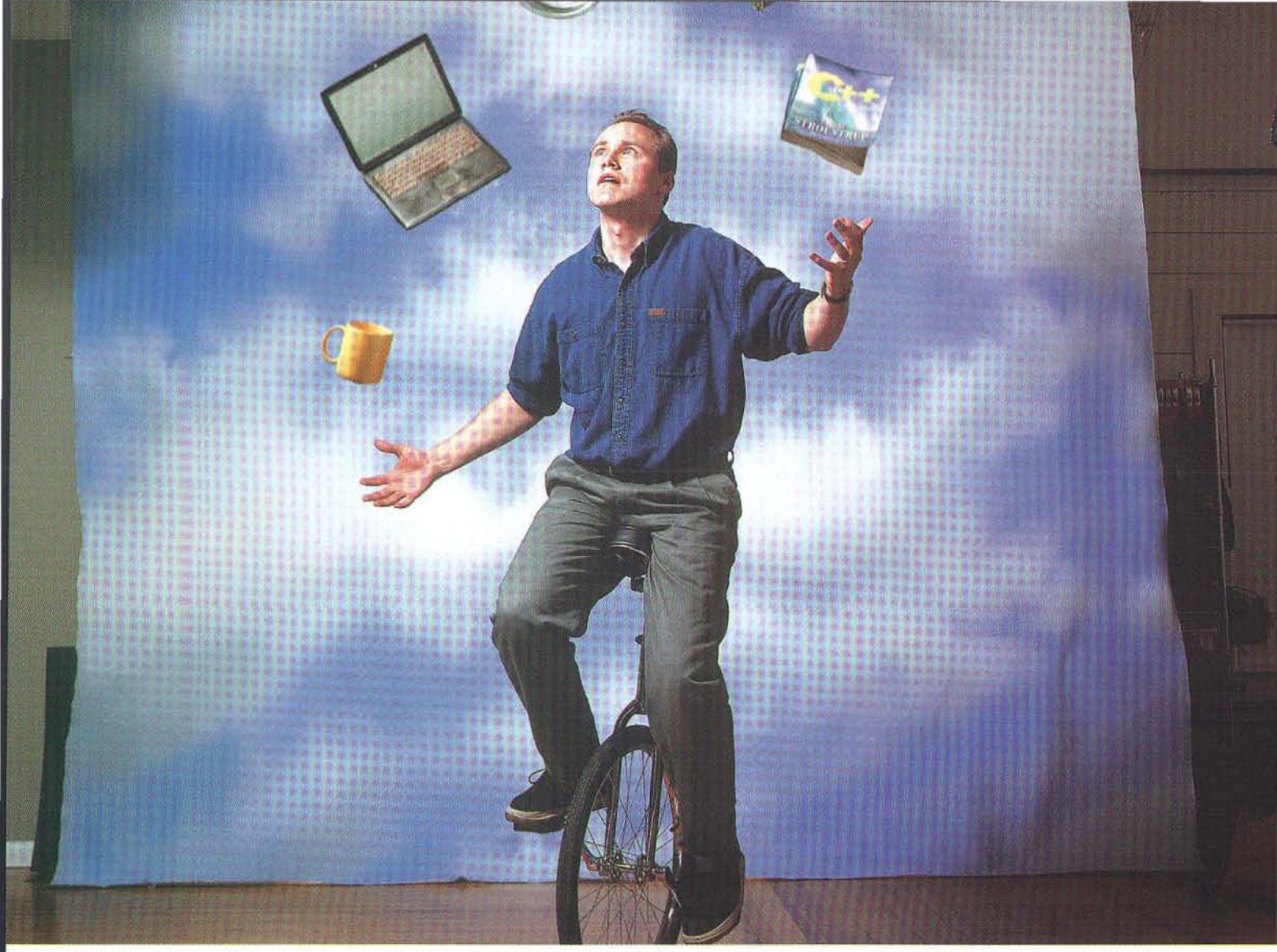
www.yellowdoglinux.com

```
  return result;
}

static BigNum SmallValue(ulong value,ulong sign)
// returns the signed BigNum version of value, 32 bits max.
{
  MyNumPtr C=New(2,sign);
  C->lastTerm=(value>0xFFFF);
  C->term[0]=value;
  C->term[1]=HI(value);
  return C->Convert2BigNum();
}

/*************************************************************
/
/                    External C Functions
/     deal with special cases, then call arithmetic if and as needed.
/
*************************************************************/

BigNum NewBigNum (              /* create a BigNum */
    char sign,                  /* +1 or -1 */
    char digits[],              /* digits to be made into a BigNum */
    long numDigits              /* number of digits */
)
{
  MyNumPtr B;
  if ((numDigits==0) || (*digits == 0))
  {
    // a leading zero will be interpreted as: the whole number == 0
    // numDigits is ignored
    B=0;
  } else
  {
    B=New((numDigits+3)/4,sign);
    B->Init(ucharPtr(digits),numDigits);
  }

  BigNum result;
  result.bigNumData=B;
  result.lengthInDigits=numDigits;
  return result;
}

long /* numDigits */ BigNumToDigits( /* convert a bigNum to decimal digits
*/
    BigNum bigNumA,             /* bigNum to be converted to decimal digits
0-9 */
    char *sign,                 /* return +1 or -1 */
    char digits[]               /* decimal digits of bigNumA */
   /* storage for digits preallocated based on bigNumA.lengthInDigits */
)
{
  const MyNumPtr A=MyNumPtr(bigNumA.bigNumData);
  if (A==0)// BigNum has the special value 0
  {
    *sign=+1;
    digits[0]=0;
    return 1;
  } else
  {
    const MyNumPtr B=BinaryTo10k(bigNumA);// temporary number
    *sign = (B->SignBit()?-1:+1);
    long numDigits=B->k10ToDigits(ucharPtr(digits));
    delete [] B;
    return numDigits;
  }
}

void DisposeBigNum (        /* dispose of a BigNum */
    BigNum theBigNum            /* the BigNum to be disposed of */
)
{
  if (theBigNum.bigNumData)
  {
    delete [] theBigNum.bigNumData;
    theBigNum.bigNumData=0;
  }// else don't delete if null (BigNum had the special value of 0)
}

BigNum AddBigNums (             /* sum two BigNums, returning a new one */
    BigNum bigNumA,             /* return the sum A+B */
    BigNum bigNumB
)
{
  const MyNumPtr A=MyNumPtr(bigNumA.bigNumData);
  const MyNumPtr B=MyNumPtr(bigNumB.bigNumData);

  if (A==0)
  {
    if (B)
      return Copy(bigNumB,B->SignBit()); // 0 + B = B
    else
      return Zero(1);               // 0 + 0 = 0
  } else if (B==0)
    return Copy(bigNumA,A->SignBit());    // A + 0 = A

  int selection=
    8*(1+A->CompAbsolute(*B)) +
    4*A->IsNegative() +
    2*B->IsNegative() + kAddMode;

  MyNumPtr C=AddSubtract(*A,*B,selection);  // general case
  if (C) return C->Convert2BigNum();
  else return Zero(1);
}

BigNum SubtractBigNums ( /* subtract two BigNums, returning a new one */
    BigNum bigNumA,             /* return the difference A-B */
    BigNum bigNumB
)
{
  const MyNumPtr A=MyNumPtr(bigNumA.bigNumData);
  const MyNumPtr B=MyNumPtr(bigNumB.bigNumData);

  if (A==0)
  {
    if (B)
      return Copy(bigNumB,kNeg ^ B->SignBit());// 0 - B = -B
    else
      return Zero(1);               // 0 - 0 = 0
  } else if (B==0)
    return Copy(bigNumA,A->SignBit());      // A - 0 = A

  int selection=
    8*(1+A->CompAbsolute(*B)) +
    4*A->IsNegative() +
    2*B->IsNegative() + kSubMode;

  MyNumPtr C=AddSubtract(*A,*B,selection);              // general
case
  if (C) return C->Convert2BigNum();
  else return Zero(1);
}

BigNum MultiplyBigNums ( /* multiply two BigNums, returning a new one */
    BigNum bigNumA,             /* return the product A*B */
    BigNum bigNumB
)
{
  const MyNumPtr A=MyNumPtr(bigNumA.bigNumData);
  const MyNumPtr B=MyNumPtr(bigNumB.bigNumData);

  if ((A==0) || (B==0))     // A*B = 0 if either A or B == 0
    return Zero(1);

  const ulong sign =        // A*B is negative if sign(A) != sign(B)
    A->SignBit() ^ B->SignBit();

  MyNumPtr C=MultiplyMyNum(*A,*B,sign);// general case

  return C->Convert2BigNum();
}

BigNum DivideBigNums (      /* divide two BigNums, returning a new one */
    BigNum bigNumA,             /* return the quotient A/B, discarding the
remainder */
    BigNum bigNumB
)
{
  const MyNumPtr A=MyNumPtr(bigNumA.bigNumData);
  const MyNumPtr B=MyNumPtr(bigNumB.bigNumData);

  if (B==0)             // A/B is an error if B==0
    return Zero(-1);
  if (A==0)             // A/B = 0 if A==0
```

```
    return Zero(1);

  int comparison=A->CompAbsolute(*B);
  if (comparison < 0)      //A/B = 0 if (A < B)
    return Zero(1);

  const ulong sign =       //A/B is negative if sign(A) != sign(B)
    A->SignBit() ^ B->SignBit();

  if (comparison == 0)     //A/B = +-1 if (A == B)
    return SmallValue(1,sign);

  MyNumPtr C=DivideMyNum(*A,*B,sign);   // general case

  return C->Convert2BigNum();
}

BigNum ModBigNums (        /* divide two BigNums, returning a new one */
    BigNum bigNumA,        /* return the remainder A%B, discarding the
quotient */
    BigNum bigNumB
)
{
  const MyNumPtr A=MyNumPtr(bigNumA.bigNumData);
  const MyNumPtr B=MyNumPtr(bigNumB.bigNumData);

  if (A==0)                //A mod B = 0 if (A==0)
    return Zero(1);

  const ulong sign=A->SignBit();
    // sign(A mod B) = sign(A), sign(B) does not matter

  if (B==0)                //A mod B = A if (B==0)
    return Copy(bigNumA.sign);

  const int comparison=A->CompAbsolute(*B);
  if (comparison == 0)     //A mod B = 0 if (A==B)
    return Zero(1);

  MyNumPtr C=(comparison < 0) ?
    NewCopy(*A,sign) :     //A mod B = A if (A<B)
    ModMyNum(*A,*B,sign); // general case

  return C->Convert2BigNum();
}

BigNum PowerBigNums (
  /* calculate one Bignum to the power of another, returning a new one */
    BigNum bigNumA,        /* return A raised to the power B */
    BigNum bigNumB
)
{
  const MyNumPtr A=MyNumPtr(bigNumA.bigNumData);
  const MyNumPtr B=MyNumPtr(bigNumB.bigNumData);

  // special cases:
  if (A==0) return Zero(1);          // 0 ** N = 0
  if (B==0) return SmallValue(1,kPos); //A ** 0 = 1

  // Need to take care of the sign:
  // sign of result is negative only if A is negative, and B is odd
  const ulong sign=(A->SignBit() & (B->IsOdd()<<31));

  if (A->IsSmall() && (A->Ulong()==1))
    return SmallValue(1,sign);       // +1 ** +B = +1
                        // -1 ** +-B = +1 (B even) or -1 (B
odd)

  // all other cases of B negative:
  if (B->IsNegative())               //A ** -B = 1/(A ** B)
    return Zero(1);                  //A ** -B is <1, hence truncated to 0

  if (!B->IsSmall())  // this means, we have an exponent that is too large
    return Zero(-1);// large powers are not supported (too much memory
implied)

  const ulong exponent=B->Ulong();
  MyNumPtr C=PowerMyNum(*A,exponent,sign);// general case

  return C->Convert2BigNum();
}

BigNum SqrtBigNum (        /* find the sqrt of a BigNum, returning a new one
*/
    BigNum bigNumA         /* return the square root of A */
)
{
  const MyNumPtr A=MyNumPtr(bigNumA.bigNumData);

  if (A==0) return Zero(1);          // sqrt(0) = 0
  if (A->IsNegative()) return Zero(-1);  // sqrt(-x) is caller's error

  if (A->IsSmall())                  // sqrt(small value) handled directly
  {
    ulong x=A->Ulong();
    return SmallValue(sqrt(x),kPos);
  }

  MyNumPtr C=SquareRootMyNum(*A);        // general case

  return C->Convert2BigNum();
}

// Additional function, provided for testing only:
int CompareBigNum(
  BigNum bigNumA,    /* returns + 0 - for bigNumA > == < bigNumB */
  BigNum bigNumB
)
{
  const MyNumPtr A=MyNumPtr(bigNumA.bigNumData);
  const MyNumPtr B=MyNumPtr(bigNumB.bigNumData);

  int result=A->CompAbsolute(*B);
  if (result==0)
    return 0;
  else
  {
    ulong asign=A->SignBit();
    ulong bsign=B->SignBit();
    if (asign==bsign)
    {
      if (asign)// both negative
      return -result;
      else return result;
    } else
    {
      if (asign) // a only negative
        return -1;
      else return 1;
    }
  }
}
```

BIGMULT.CP

```
#include <string.h>
#include "BigMult.h"

/*****************************************************************
/
/  Fast multiple precision multiplier
/  Copyright © 2000, Ernst Munter, Kanata, ON, Canada.
/
/
/  Inspired by "How fast can we multiply" in The Art of Computer Programming
/  by Donald Knuth (vol.2, page 295): Split and combine terms to reduce
/  running time from order N-squared to N**(lg 3) for large BigNums.
/
/  The routine Multiply() calls itself recursively until the size of the
/  factors is reduced to a range below 240 bits.
/
/  A seperate unrolled function takes care of the final matrix multiplication
/  for each size (minimum size is 8 by 8 terms).
/
*****************************************************************/

#define HI(chunk) ((chunk)>>16)
#define LO(chunk) ((chunk)&0xFFFF)


// We try to work the final multiplication within registers. To conserve registers,
// pairs of short factors, e.g. U[i] and U[i+1] are read into a single long register.
// Note: this technique was not implemented to be portable to little-endian CPUs.

// The strategy works perfectly for N=8 and N=9. For N>9, the compiler (CW-5.3)
// fails to accomplish this, and does save some values on the stack.
```



Programming Doesn't Have To Be This Difficult. It's REALbasic!

NEW Version! REALbasic 2.1* FREE Update!

REALbasic is the award-winning, visual, object-oriented BASIC development environment for the Macintosh.

Use REALbasic's visual interface builder and platform-independent language to build native, compiled—not interpreted—professional quality applications in a fraction of the time it would take in C/C++. Because our language is platform-independent you only need to write a single set of code to create applications for both Macintosh and Windows. Leverage your C/C++ experience to extend REALbasic's capabilities using shared libraries, Mac OS Toolbox and Win32 API calls or by writing cross-platform REALbasic plug-ins.

Create prototypes, Internet applications, database front-ends, even games with REALbasic! Its OOP language employs everything you'd expect from a modern development environment—methods, properties, classes, subclassing, inheritance, constructors and destructors, virtual methods, and more. The IDE supports multi-threading, extensibility, TCP/IP controls, plus multimedia and QuickTime tools, and support for standards such as SQL, ODBC, AppleScript, and XCMDs. You can even import Visual Basic forms and modules.

Go to www.realbasic.com NOW to download a FREE trial version or call 512.263.1233.

Come see us at MacWorld New York — Booth #1907

REALbasic®

MacPower 2000 Products of The Year

MacHome

Macworld

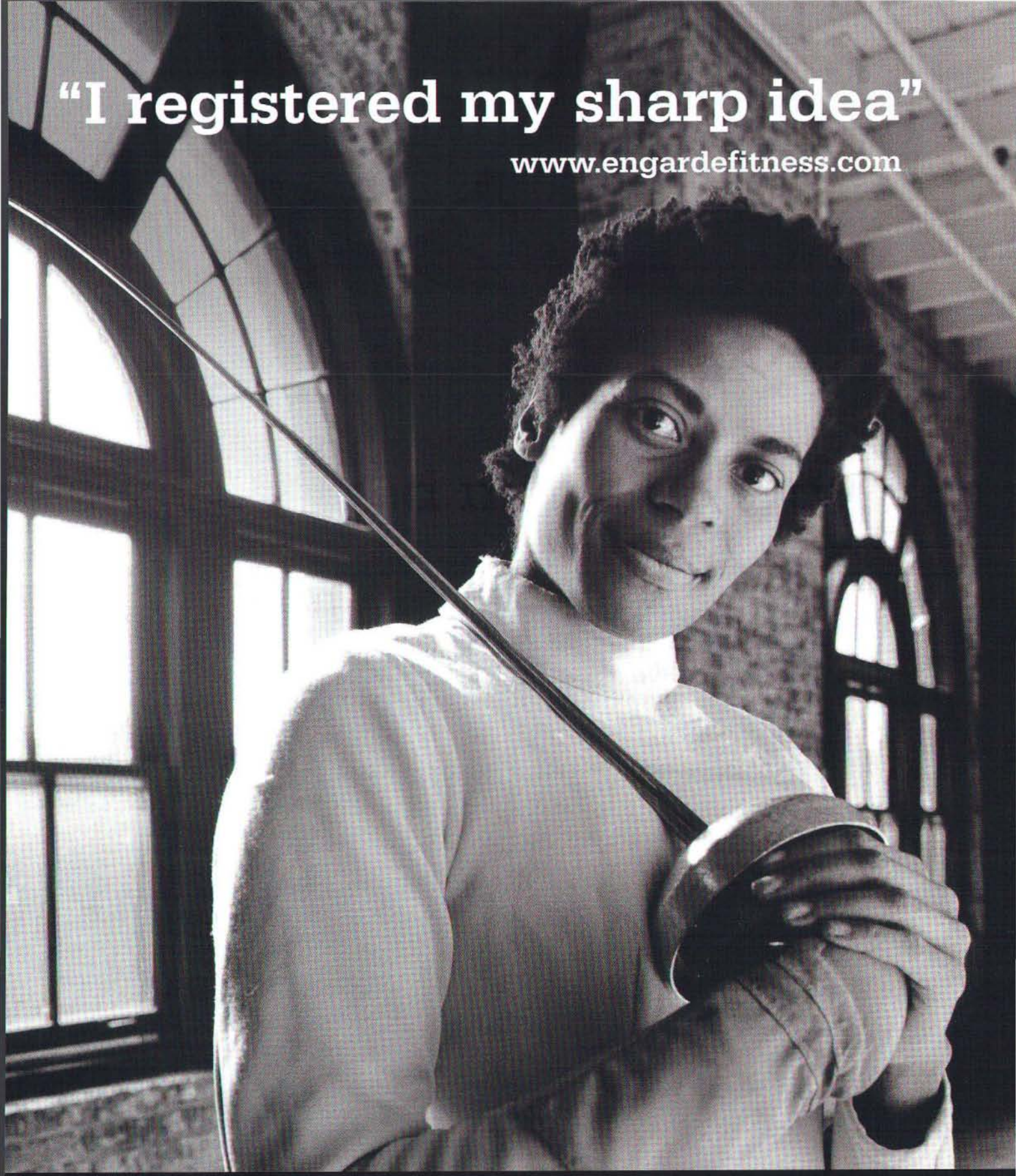
Apple Design Award

2000 Runner-Up - Best Macintosh User Experience

Mac Fan MVP '99 MOST VALUABLE PRODUCT

*Free update for all owners of REALbasic 2.0 and above. REALbasic and the REALbasic logo are trademarks of REAL Software, Inc. Apple and the Apple logo are trademarks of Apple Computer, Inc., registered in the U.S., used with permission. All other trademarks are the property of their respective owners.

```
// For N >= 12, a more straight forward technique is actually faster: we read the
// factors U[i] and V[j] from memory (cache) directly, each time they are used.

#define U0 (HI(U01))
#define U1 (LO(U01))
#define U2 (HI(U23))
#define U3 (LO(U23))
#define U4 (HI(U45))
#define U5 (LO(U45))
#define U6 (HI(U67))
#define U7 (LO(U67))
#define U8 (HI(U89))
#define U9 (LO(U89))
#define U10 (HI(U1011))
#define U11 (LO(U1011))
#define U12 (HI(U1213))
#define U13 (LO(U1213))
#define U14 (HI(U1415))
#define U15 (LO(U1415))

#define V0 (HI(V01))
#define V1 (LO(V01))
#define V2 (HI(V23))
#define V3 (LO(V23))
#define V4 (HI(V45))
#define V5 (LO(V45))
#define V6 (HI(V67))
#define V7 (LO(V67))
#define V8 (HI(V89))
#define V9 (LO(V89))
#define V10 (HI(V1011))
#define V11 (LO(V1011))
#define V12 (HI(V1213))
#define V13 (LO(V1213))
#define V14 (HI(V1415))
#define V15 (LO(V1415))
#define F(a,b) {x=LO(y) + (U##a) * (V##b);
y=HI(y)+HI(x);}
#define G(a,b) {x=LO(x) + (U##a) * (V##b);
y+=HI(x);}

static void FinalMultiply8(const ushort* U,
      const ushort* V,ushort* P)
{
  ulong U01=*((ulong*)(U+0));
  ulong U23=*((ulong*)(U+2));
  ulong U45=*((ulong*)(U+4));
  ulong U67=*((ulong*)(U+6));

  ulong V01=*((ulong*)(V+0));
  ulong V23=*((ulong*)(V+2));
  ulong V45=*((ulong*)(V+4));
  ulong V67=*((ulong*)(V+6));

  ulong x=U0*V0, y=HI(x);
  P[0]=x;
  F(0,1); G(1,0);  P[1]=x;
  F(0,2); G(1,1); G(2,0); P[2]=x;
  F(0,3); G(1,2); G(2,1); G(3,0); P[3]=x;
  F(0,4); G(1,3); G(2,2); G(3,1); G(4,0); P[4]=x;
  F(0,5); G(1,4); G(2,3); G(3,2); G(4,1); G(5,0);
P[5]=x;
  F(0,6); G(1,5); G(2,4); G(3,3); G(4,2); G(5,1);
G(6,0); P[6]=x;
  F(0,7); G(1,6); G(2,5); G(3,4); G(4,3); G(5,2);
G(6,1); G(7,0);
      P[7]=x;
  F(1,7); G(2,6); G(3,5); G(4,4); G(5,3); G(6,2);
G(7,1); P[8]=x;
  F(2,7); G(3,6); G(4,5); G(5,4); G(6,3); G(7,2);
P[9]=x;
  F(3,7); G(4,6); G(5,5); G(6,4); G(7,3); P[10]=x;
  F(4,7); G(5,6); G(6,5); G(7,4); P[11]=x;
  F(5,7); G(6,6); G(7,5); P[12]=x;
  F(6,7); G(7,6); P[13]=x;
  F(7,7); P[14]=x;
  P[15]=y;
}

static void FinalMultiply9(const ushort* U,
      const ushort* V,ushort* P)
{
  ulong U01=*((ulong*)(U+0));
  ulong U23=*((ulong*)(U+2));
  ulong U45=*((ulong*)(U+4));
  ulong U67=*((ulong*)(U+6));
  ulong U89=*((ulong*)(U+8));

  ulong V01=*((ulong*)(V+0));
  ulong V23=*((ulong*)(V+2));
  ulong V45=*((ulong*)(V+4));
  ulong V67=*((ulong*)(V+6));
  ulong V89=*((ulong*)(V+8));

  ulong x=U0*V0, y=HI(x);
  P[0]=x;
  F(0,1); G(1,0);  P[1]=x;
  F(0,2); G(1,1); G(2,0); P[2]=x;
  F(0,3); G(1,2); G(2,1); G(3,0); P[3]=x;
  F(0,4); G(1,3); G(2,2); G(3,1); G(4,0); P[4]=x;
  F(0,5); G(1,4); G(2,3); G(3,2); G(4,1); G(5,0);
P[5]=x;
  F(0,6); G(1,5); G(2,4); G(3,3); G(4,2); G(5,1);
G(6,0); P[6]=x;
  F(0,7); G(1,6); G(2,5); G(3,4); G(4,3); G(5,2);
G(6,1); G(7,0);
      P[7]=x;
  F(0,8); G(1,7); G(2,6); G(3,5); G(4,4); G(5,3);
G(6,2); G(7,1);
      G(8,0); P[8]=x;
  F(1,8); G(2,7); G(3,6); G(4,5); G(5,4); G(6,3);
G(7,2); G(8,1);
      P[9]=x;
  F(2,8); G(3,7); G(4,6); G(5,5); G(6,4); G(7,3);
G(8,2);
      P[10]=x;
  F(3,8); G(4,7); G(5,6); G(6,5); G(7,4); G(8,3);
P[11]=x;
  F(4,8); G(5,7); G(6,6); G(7,5); G(8,4); P[12]=x;
  F(5,8); G(6,7); G(7,6); G(8,5); P[13]=x;
  F(6,8); G(7,7); G(8,6); P[14]=x;
  F(7,8); G(8,7); P[15]=x;
  F(8,8); P[16]=x;
  P[17]=y;
}

static void FinalMultiply10(const ushort* U,
      const ushort* V,ushort* P)
{
  ulong U01=*((ulong*)(U+0));
  ulong U23=*((ulong*)(U+2));
  ulong U45=*((ulong*)(U+4));
  ulong U67=*((ulong*)(U+6));
  ulong U89=*((ulong*)(U+8));

  ulong V01=*((ulong*)(V+0));
  ulong V23=*((ulong*)(V+2));
  ulong V45=*((ulong*)(V+4));
  ulong V67=*((ulong*)(V+6));
  ulong V89=*((ulong*)(V+8));

  ulong x=U0*V0, y=HI(x);
  P[0]=x;
  F(0,1); G(1,0);  P[1]=x;
  F(0,2); G(1,1); G(2,0); P[2]=x;
  F(0,3); G(1,2); G(2,1); G(3,0); P[3]=x;
  F(0,4); G(1,3); G(2,2); G(3,1); G(4,0); P[4]=x;
  F(0,5); G(1,4); G(2,3); G(3,2); G(4,1); G(5,0);
P[5]=x;
  F(0,6); G(1,5); G(2,4); G(3,3); G(4,2); G(5,1);
G(6,0); P[6]=x;
  F(0,7); G(1,6); G(2,5); G(3,4); G(4,3); G(5,2);
G(6,1); G(7,0);
      P[7]=x;
  F(0,8); G(1,7); G(2,6); G(3,5); G(4,4); G(5,3);
G(6,2); G(7,1);
      G(8,0); P[8]=x;
  F(0,9); G(1,8); G(2,7); G(3,6); G(4,5); G(5,4);
G(6,3); G(7,2);
      G(8,1); G(9,0); P[9]=x;
  F(1,9); G(2,8); G(3,7); G(4,6); G(5,5); G(6,4);
G(7,3); G(8,2);
      G(9,1); P[10]=x;
  F(2,9); G(3,8); G(4,7); G(5,6); G(6,5); G(7,4);
```



Sharon Monplaisir, Olympic fencer and entrepreneur, registered her domain name to put her business online at register.com. Visit us at www.register.com or call us at 1-800-7-WWW-NET, and we'll help you register your domain name right over the phone.

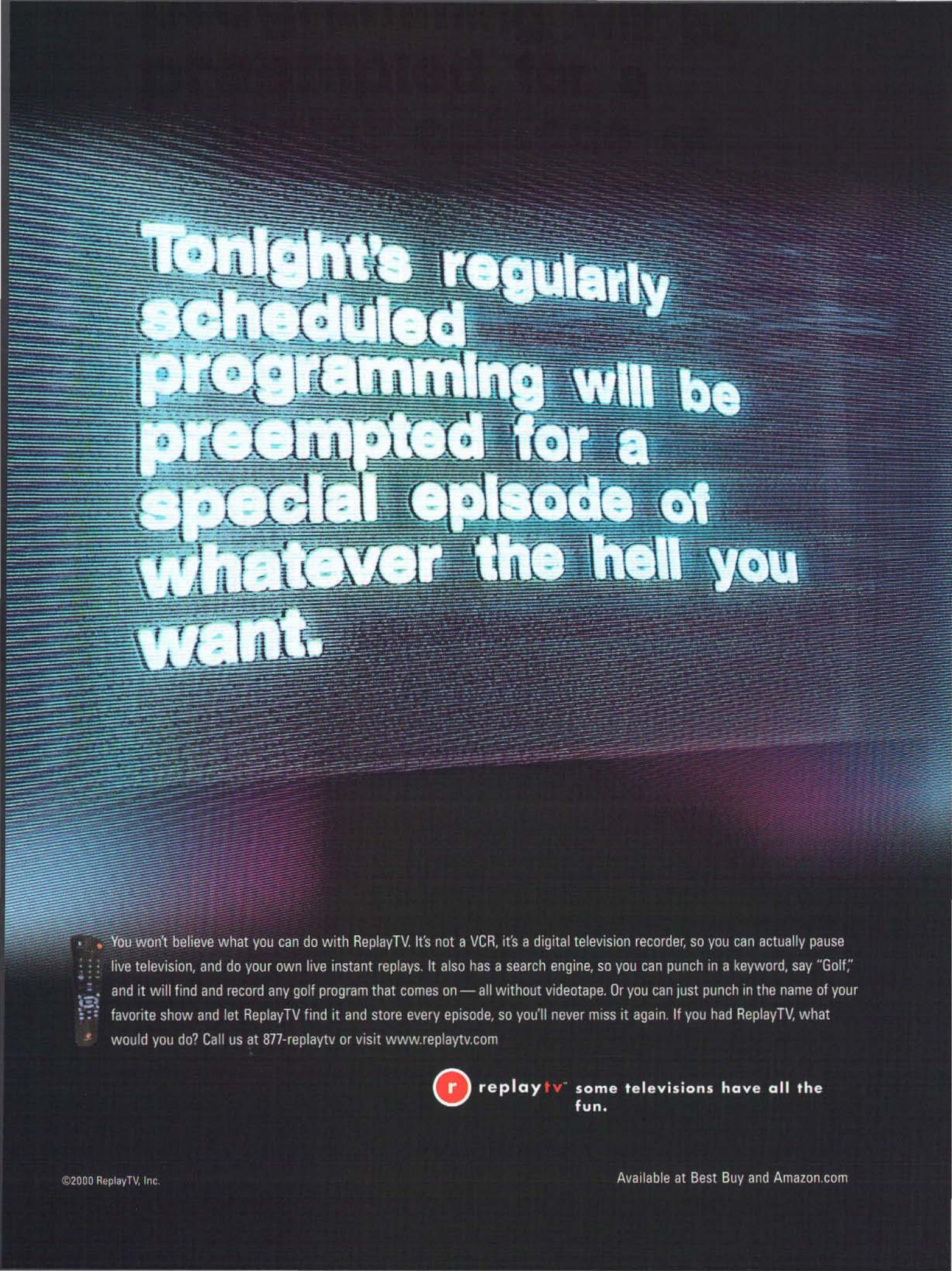
register.com™

the first step on the web℠

```
G(8,3); G(9,2);
      P[11]=x;
  F(3,9); G(4,8); G(5,7); G(6,6); G(7,5); G(8,4);
G(9,3);
      P[12]=x;
  F(4,9); G(5,8); G(6,7); G(7,6); G(8,5); G(9,4);
P[13]=x;
  F(5,9); G(6,8); G(7,7); G(8,6); G(9,5); P[14]=x;
  F(6,9); G(7,8); G(8,7); G(9,6); P[15]=x;
  F(7,9); G(8,8); G(9,7); P[16]=x;
  F(8,9); G(9,8); P[17]=x;
  F(9,9); P[18]=x;
  P[19]=y;
}

static void FinalMultiply11(const ushort* U,
      const ushort* V,ushort* P)
{
  ulong U01=*((ulong*)(U+0));
  ulong U23=*((ulong*)(U+2));
  ulong U45=*((ulong*)(U+4));
  ulong U67=*((ulong*)(U+6));
  ulong U89=*((ulong*)(U+8));
  ulong U1011=*((ulong*)(U+10));

  ulong V01=*((ulong*)(V+0));
  ulong V23=*((ulong*)(V+2));
  ulong V45=*((ulong*)(V+4));
  ulong V67=*((ulong*)(V+6));
  ulong V89=*((ulong*)(V+8));
  ulong V1011=*((ulong*)(V+10));

  ulong x=U0*V0, y=HI(x);
  P[0]=x;
  F(0,1); G(1,0);  P[1]=x;
  F(0,2); G(1,1); G(2,0); P[2]=x;
  F(0,3); G(1,2); G(2,1); G(3,0); P[3]=x;
  F(0,4); G(1,3); G(2,2); G(3,1); G(4,0); P[4]=x;
  F(0,5); G(1,4); G(2,3); G(3,2); G(4,1); G(5,0);
P[5]=x;
  F(0,6); G(1,5); G(2,4); G(3,3); G(4,2); G(5,1);
G(6,0); P[6]=x;
  F(0,7); G(1,6); G(2,5); G(3,4); G(4,3); G(5,2);
G(6,1); G(7,0);
      P[7]=x;
  F(0,8); G(1,7); G(2,6); G(3,5); G(4,4); G(5,3);
G(6,2); G(7,1);
      G(8,0); P[8]=x;
  F(0,9); G(1,8); G(2,7); G(3,6); G(4,5); G(5,4);
G(6,3); G(7,2);
      G(8,1); G(9,0); P[9]=x;
  F(0,10); G(1,9); G(2,8); G(3,7); G(4,6); G(5,5);
G(6,4);
      G(7,3); G(8,2); G(9,1); G(10,0); P[10]=x;
  F(1,10); G(2,9); G(3,8); G(4,7); G(5,6); G(6,5);
G(7,4);
      G(8,3); G(9,2); G(10,1); P[11]=x;
  F(2,10); G(3,9); G(4,8); G(5,7); G(6,6); G(7,5);
G(8,4);
      G(9,3); G(10,2); P[12]=x;
  F(3,10); G(4,9); G(5,8); G(6,7); G(7,6); G(8,5);
G(9,4);
      G(10,3); P[13]=x;
  F(4,10); G(5,9); G(6,8); G(7,7); G(8,6); G(9,5);
G(10,4);
       P[14]=x;
  F(5,10); G(6,9); G(7,8); G(8,7); G(9,6); G(10,5);
P[15]=x;
  F(6,10); G(7,9); G(8,8); G(9,7); G(10,6);
P[16]=x;
  F(7,10); G(8,9); G(9,8); G(10,7); P[17]=x;
  F(8,10); G(9,9); G(10,8); P[18]=x;
  F(9,10); G(10,9); P[19]=x;
  F(10,10); P[20]=x;
  P[21]=y;
}
```

// Macros redefined to read factors directly from memory

```
#undef F
#undef G
#define F(a,b) {x=LO(y) + (U[a]) * (V[b]);
y=HI(y)+HI(x);}

#define G(a,b) {x=LO(x) + (U[a]) * (V[b]);
y+=HI(x);}

static void FinalMultiply12(const ushort* U,
      const ushort* V,ushort* P)
{
  ulong x=U[0]*V[0], y=HI(x);
  P[0]=x;
  F(0,1); G(1,0);  P[1]=x;
  F(0,2); G(1,1); G(2,0); P[2]=x;
  F(0,3); G(1,2); G(2,1); G(3,0); P[3]=x;
  F(0,4); G(1,3); G(2,2); G(3,1); G(4,0); P[4]=x;
  F(0,5); G(1,4); G(2,3); G(3,2); G(4,1); G(5,0);
P[5]=x;
  F(0,6); G(1,5); G(2,4); G(3,3); G(4,2); G(5,1);
G(6,0); P[6]=x;
  F(0,7); G(1,6); G(2,5); G(3,4); G(4,3); G(5,2);
G(6,1); G(7,0);
      P[7]=x;
  F(0,8); G(1,7); G(2,6); G(3,5); G(4,4); G(5,3);
G(6,2); G(7,1);
      G(8,0); P[8]=x;
  F(0,9); G(1,8); G(2,7); G(3,6); G(4,5); G(5,4);
G(6,3); G(7,2);
      G(8,1); G(9,0); P[9]=x;
  F(0,10); G(1,9); G(2,8); G(3,7); G(4,6); G(5,5);
G(6,4);
      G(7,3); G(8,2); G(9,1); G(10,0); P[10]=x;
  F(0,11); G(1,10); G(2,9); G(3,8); G(4,7); G(5,6);
G(6,5);
      G(7,4); G(8,3); G(9,2); G(10,1); G(11,0);
P[11]=x;
  F(1,11); G(2,10); G(3,9); G(4,8); G(5,7); G(6,6);
G(7,5);
      G(8,4); G(9,3); G(10,2); G(11,1); P[12]=x;
  F(2,11); G(3,10); G(4,9); G(5,8); G(6,7); G(7,6);
G(8,5);
      G(9,4); G(10,3); G(11,2); P[13]=x;
  F(3,11); G(4,10); G(5,9); G(6,8); G(7,7); G(8,6);
G(9,5);
      G(10,4); G(11,3); P[14]=x;
  F(4,11); G(5,10); G(6,9); G(7,8); G(8,7); G(9,6);
G(10,5);
      G(11,4); P[15]=x;
  F(5,11); G(6,10); G(7,9); G(8,8); G(9,7);
G(10,6); G(11,5);
      P[16]=x;
  F(6,11); G(7,10); G(8,9); G(9,8); G(10,7);
G(11,6); P[17]=x;
  F(7,11); G(8,10); G(9,9); G(10,8); G(11,7);
P[18]=x;
  F(8,11); G(9,10); G(10,9); G(11,8); P[19]=x;
  F(9,11); G(10,10); G(11,9); P[20]=x;
  F(10,11); G(11,10); P[21]=x;
  F(11,11); P[22]=x;
  P[23]=y;
}

static void FinalMultiply13(const ushort* U,
      const ushort* V,ushort* P)
{
  ulong x=U[0]*V[0], y=HI(x);
  P[0]=x;
  F(0,1); G(1,0);  P[1]=x;
  F(0,2); G(1,1); G(2,0); P[2]=x;
  F(0,3); G(1,2); G(2,1); G(3,0); P[3]=x;
  F(0,4); G(1,3); G(2,2); G(3,1); G(4,0); P[4]=x;
  F(0,5); G(1,4); G(2,3); G(3,2); G(4,1); G(5,0);
P[5]=x;
  F(0,6); G(1,5); G(2,4); G(3,3); G(4,2); G(5,1);
G(6,0); P[6]=x;
  F(0,7); G(1,6); G(2,5); G(3,4); G(4,3); G(5,2);
G(6,1); G(7,0);
      P[7]=x;
  F(0,8); G(1,7); G(2,6); G(3,5); G(4,4); G(5,3);
G(6,2); G(7,1);
      G(8,0); P[8]=x;
  F(0,9); G(1,8); G(2,7); G(3,6); G(4,5); G(5,4);
G(6,3); G(7,2);
      G(8,1); G(9,0); P[9]=x;
  F(0,10); G(1,9); G(2,8); G(3,7); G(4,6); G(5,5);
G(6,4);
```



You won't believe what you can do with ReplayTV. It's not a VCR, it's a digital television recorder, so you can actually pause live television, and do your own live instant replays. It also has a search engine, so you can punch in a keyword, say "Golf," and it will find and record any golf program that comes on — all without videotape. Or you can just punch in the name of your favorite show and let ReplayTV find it and store every episode, so you'll never miss it again. If you had ReplayTV, what would you do? Call us at 877-replaytv or visit www.replaytv.com

replaytv™ some televisions have all the fun.

Available at Best Buy and Amazon.com

```
      G(7,3); G(8,2); G(9,1); G(10,0); P[10]=x;
  F(0,11); G(1,10); G(2,9); G(3,8); G(4,7); G(5,6);
G(6,5);
      G(7,4); G(8,3); G(9,2); G(10,1); G(11,0);
P[11]=x;
  F(0,12); G(1,11); G(2,10); G(3,9); G(4,8);
G(5,7); G(6,6);
      G(7,5); G(8,4); G(9,3); G(10,2); G(11,1);
G(12,0); P[12]=x;
  F(1,12); G(2,11); G(3,10); G(4,9); G(5,8);
G(6,7); G(7,6);
      G(8,5); G(9,4); G(10,3); G(11,2); G(12,1);
P[13]=x;
  F(2,12); G(3,11); G(4,10); G(5,9); G(6,8);
G(7,7); G(8,6);
      G(9,5); G(10,4); G(11,3); G(12,2); P[14]=x;
  F(3,12); G(4,11); G(5,10); G(6,9); G(7,8);
G(8,7); G(9,6);
      G(10,5); G(11,4); G(12,3); P[15]=x;
  F(4,12); G(5,11); G(6,10); G(7,9); G(8,8);
G(9,7); G(10,6);
      G(11,5); G(12,4); P[16]=x;
  F(5,12); G(6,11); G(7,10); G(8,9); G(9,8);
G(10,7); G(11,6);
      G(12,5); P[17]=x;
  F(6,12); G(7,11); G(8,10); G(9,9); G(10,8);
G(11,7); G(12,6);
      P[18]=x;
  F(7,12); G(8,11); G(9,10); G(10,9); G(11,8);
G(12,7); P[19]=x;
  F(8,12); G(9,11); G(10,10); G(11,9); G(12,8);
P[20]=x;
  F(9,12); G(10,11); G(11,10); G(12,9); P[21]=x;
  F(10,12); G(11,11); G(12,10); P[22]=x;
  F(11,12); G(12,11); P[23]=x;
  F(12,12); P[24]=x;
  P[25]=y;
}

static void FinalMultiply14(const ushort* U,
      const ushort* V,ushort* P)
{
  ulong x=U[0]*V[0], y=HI(x);
  P[0]=x;
  F(0,1); G(1,0);  P[1]=x;
  F(0,2); G(1,1); G(2,0); P[2]=x;
  F(0,3); G(1,2); G(2,1); G(3,0); P[3]=x;
  F(0,4); G(1,3); G(2,2); G(3,1); G(4,0); P[4]=x;
  F(0,5); G(1,4); G(2,3); G(3,2); G(4,1); G(5,0);
P[5]=x;
  F(0,6); G(1,5); G(2,4); G(3,3); G(4,2); G(5,1);
G(6,0); P[6]=x;
  F(0,7); G(1,6); G(2,5); G(3,4); G(4,3); G(5,2);
G(6,1); G(7,0);
      P[7]=x;
  F(0,8); G(1,7); G(2,6); G(3,5); G(4,4); G(5,3);
G(6,2); G(7,1);
      G(8,0); P[8]=x;
  F(0,9); G(1,8); G(2,7); G(3,6); G(4,5); G(5,4);
G(6,3); G(7,2);
      G(8,1); G(9,0); P[9]=x;
  F(0,10); G(1,9); G(2,8); G(3,7); G(4,6); G(5,5);
G(6,4);
      G(7,3); G(8,2); G(9,1); G(10,0); P[10]=x;
  F(0,11); G(1,10); G(2,9); G(3,8); G(4,7); G(5,6);
G(6,5);
      G(7,4); G(8,3); G(9,2); G(10,1); G(11,0);
P[11]=x;
  F(0,12); G(1,11); G(2,10); G(3,9); G(4,8);
G(5,7); G(6,6);
      G(7,5); G(8,4); G(9,3); G(10,2); G(11,1);
G(12,0); P[12]=x;
  F(0,13); G(1,12); G(2,11); G(3,10); G(4,9);
G(5,8); G(6,7);
      G(7,6); G(8,5); G(9,4); G(10,3); G(11,2);
G(12,1); G(13,0);
      P[13]=x;
  F(1,13); G(2,12); G(3,11); G(4,10); G(5,9);
G(6,8); G(7,7);
      G(8,6); G(9,5); G(10,4); G(11,3); G(12,2);
G(13,1); P[14]=x;
  F(2,13); G(3,12); G(4,11); G(5,10); G(6,9);
G(7,8); G(8,7);
      G(9,6); G(10,5); G(11,4); G(12,3); G(13,2);
P[15]=x;
  F(3,13); G(4,12); G(5,11); G(6,10); G(7,9);
G(8,8); G(9,7);
      G(10,6); G(11,5); G(12,4); G(13,3); P[16]=x;
  F(4,13); G(5,12); G(6,11); G(7,10); G(8,9);
G(9,8); G(10,7);
      G(11,6); G(12,5); G(13,4); P[17]=x;
  F(5,13); G(6,12); G(7,11); G(8,10); G(9,9);
G(10,8); G(11,7);
      G(12,6); G(13,5); P[18]=x;
  F(6,13); G(7,12); G(8,11); G(9,10); G(10,9);
G(11,8); G(12,7);
      G(13,6); P[19]=x;
  F(7,13); G(8,12); G(9,11); G(10,10); G(11,9);
G(12,8); G(13,7);
      P[20]=x;
  F(8,13); G(9,12); G(10,11); G(11,10); G(12,9);
G(13,8);
      P[21]=x;
  F(9,13); G(10,12); G(11,11); G(12,10); G(13,9);
P[22]=x;
  F(10,13); G(11,12); G(12,11); G(13,10); P[23]=x;
  F(11,13); G(12,12); G(13,11); P[24]=x;
  F(12,13); G(13,12); P[25]=x;
  F(13,13); P[26]=x;
  P[27]=y;
}

static void FinalMultiply15(const ushort* U,
      const ushort* V,ushort* P)
{
  ulong x=U[0]*V[0], y=HI(x);
  P[0]=x;
  F(0,1); G(1,0);  P[1]=x;
  F(0,2); G(1,1); G(2,0); P[2]=x;
  F(0,3); G(1,2); G(2,1); G(3,0); P[3]=x;
  F(0,4); G(1,3); G(2,2); G(3,1); G(4,0); P[4]=x;
  F(0,5); G(1,4); G(2,3); G(3,2); G(4,1); G(5,0);
P[5]=x;
  F(0,6); G(1,5); G(2,4); G(3,3); G(4,2); G(5,1);
G(6,0); P[6]=x;
  F(0,7); G(1,6); G(2,5); G(3,4); G(4,3); G(5,2);
G(6,1); G(7,0);
      P[7]=x;
  F(0,8); G(1,7); G(2,6); G(3,5); G(4,4); G(5,3);
G(6,2); G(7,1);
      G(8,0); P[8]=x;
  F(0,9); G(1,8); G(2,7); G(3,6); G(4,5); G(5,4);
G(6,3); G(7,2);
      G(8,1); G(9,0); P[9]=x;
  F(0,10); G(1,9); G(2,8); G(3,7); G(4,6); G(5,5);
G(6,4);
      G(7,3); G(8,2); G(9,1); G(10,0); P[10]=x;
  F(0,11); G(1,10); G(2,9); G(3,8); G(4,7); G(5,6);
G(6,5);
      G(7,4); G(8,3); G(9,2); G(10,1); G(11,0);
P[11]=x;
  F(0,12); G(1,11); G(2,10); G(3,9); G(4,8);
G(5,7); G(6,6);
      G(7,5); G(8,4); G(9,3); G(10,2); G(11,1);
G(12,0); P[12]=x;
  F(0,13); G(1,12); G(2,11); G(3,10); G(4,9);
G(5,8); G(6,7);
      G(7,6); G(8,5); G(9,4); G(10,3); G(11,2);
G(12,1); G(13,0);
      P[13]=x;
  F(0,14); G(1,13); G(2,12); G(3,11); G(4,10);
G(5,9); G(6,8);
      G(7,7); G(8,6); G(9,5); G(10,4); G(11,3);
G(12,2); G(13,1);
      G(14,0); P[14]=x;
  F(1,14); G(2,13); G(3,12); G(4,11); G(5,10);
G(6,9); G(7,8);
      G(8,7); G(9,6); G(10,5); G(11,4); G(12,3);
G(13,2); G(14,1);
```

```
      P[15]=x;
  F(2,14); G(3,13); G(4,12); G(5,11); G(6,10);
G(7,9); G(8,8);
      G(9,7); G(10,6); G(11,5); G(12,4); G(13,3);
G(14,2);
      P[16]=x;
  F(3,14); G(4,13); G(5,12); G(6,11); G(7,10);
G(8,9); G(9,8);
      G(10,7); G(11,6); G(12,5); G(13,4); G(14,3);
P[17]=x;
  F(4,14); G(5,13); G(6,12); G(7,11); G(8,10);
G(9,9); G(10,8);
      G(11,7); G(12,6); G(13,5); G(14,4); P[18]=x;
  F(5,14); G(6,13); G(7,12); G(8,11); G(9,10);
G(10,9); G(11,8);
      G(12,7); G(13,6); G(14,5); P[19]=x;
  F(6,14); G(7,13); G(8,12); G(9,11); G(10,10);
G(11,9); G(12,8);
      G(13,7); G(14,6); P[20]=x;
  F(7,14); G(8,13); G(9,12); G(10,11); G(11,10);
G(12,9);
      G(13,8); G(14,7); P[21]=x;
  F(8,14); G(9,13); G(10,12); G(11,11); G(12,10);
G(13,9);
      G(14,8); P[22]=x;
  F(9,14); G(10,13); G(11,12); G(12,11); G(13,10);
G(14,9);
      P[23]=x;
  F(10,14); G(11,13); G(12,12); G(13,11); G(14,10);
P[24]=x;
  F(11,14); G(12,13); G(13,12); G(14,11); P[25]=x;
  F(12,14); G(13,13); G(14,12); P[26]=x;
  F(13,14); G(14,13); P[27]=x;
  F(14,14); P[28]=x;
  P[29]=y;
}

static int Subtract(const ushort* A,const ushort*
B,
      ushort* D,int N)
// returns 1 if A < B
// computes difference of |A - B| into space at D
{
  int sign=0;
  for (int i=N;i>0;)
  {
    i--;
    if (A[i] < B[i])
    {
      sign=1;break;
    } else if (A[i] > B[i])
    {
      sign=0;break;
    }
  }
  if (sign) {const ushort* T=A;A=B;B=T;}

  long L=*A-*B,borrow=HI(L);
  *D=L;
  for (int i=1;i<N;i++)
  {
    L = *++A - *++B + borrow;
    borrow=HI(L);
    *++D = L;
  }
  return sign;
}

static void Accumulate(ushort* dest,const ushort*
A,int N)
// accumulates the sum of dest + A into the space at dest
{
  long L=*dest+*A,carry=HI(L);
  *dest=L;
  for (int i=1;i<N;i++)
  {
    L = *++dest + *++A + carry;
    carry=HI(L);
    *dest = L;
  }
```

```
  while (carry)
  {
    L = *++dest + carry;
    carry=HI(L);
    *dest = L;
  }
}

static void NegAccumulate(ushort* dest,const
ushort* A,int N)
// accumulates the difference of dest - A into the space at dest
{
  long L=*dest-*A,borrow=HI(L);
  *dest=L;
  for (int i=1;i<N;i++)
  {
    L = *++dest - *++A + borrow;
    borrow=HI(L);
    *dest = L;
  }
  while (borrow)
  {
    L = *++dest + borrow;
    borrow=HI(L);
    *dest = L;
  }
}

typedef void FinalMultiplyFun(const ushort* ,
    const ushort*, ushort*);

static void Dummy(const ushort*,const
ushort*,ushort*){}

static FinalMultiplyFun *FinalMultiply[16] = {
  Dummy,Dummy,Dummy,Dummy,Dummy,Dummy,Dummy,Dummy,
  FinalMultiply8,
  FinalMultiply9,
  FinalMultiply10,
  FinalMultiply11,
  FinalMultiply12,
  FinalMultiply13,
  FinalMultiply14,
  FinalMultiply15
};

void Multiply(const ushort* U,const ushort*
V,ushort* P,const int N,const long nv)
// Recursive function to multiply arrays U and V, where V may have
many leading 0s
// N is the size of U and V.  P is expected to have size 2N
// nv is the actually used size of V.
// If nv is less than 1/2N, computation of 0-value higher terms is
avoided
{
  if (N<=15)  // actually, N should always be >= 8
  {
    FinalMultiply[N](U,V,P);
    return;
  }

  memset(P,0,2*N*sizeof(ushort));
  if (nv>N/2)
  {
    const ushort* u0=U;
    const ushort* u1=U+N/2;
    const ushort* v0=V;
    const ushort* v1=V+N/2;
    ushort* p0=P;
    ushort* p1=P+N/2;
    ushort* p2=P+N;
    ushort* temp0=P+2*N;
    ushort* temp1=temp0+N/2;
    ushort* temp2=temp0+N;
// Main level of recursion;  each level results in 3 calls to Multiply at a
deeper level.
// the idea is:
// P = U * V = (u1,u0) * (v1,v0)
//     where U is split into high order terms (u1) and low order terms
(u0):
//     P = u1*v1, u1*v0 + u0*v1, u0*v0.
// The terms can be rearranged, so that only 3 instead of 4
multiplications are needed:
//     P = u1*v1, u1*v1 + (u1-u0) * (v0-v1) + u0*v0, u0*v0
// The downside is that products must be computed into temporary
space and copied
//     but the upside is that, with recursion, much fewer operations are
carried out.
//     Running time is proportional to N ** 1.58 for square problems
//       (size(U)=size(V)=N)
    Multiply(u0,v0,temp2,N/2,nv);

    Accumulate(p0,temp2,N);
    Accumulate(p1,temp2,N);

    Multiply(u1,v1,temp2,N/2,nv);
    Accumulate(p1,temp2,N);
    Accumulate(p2,temp2,N);

    int negative=Subtract(u1,u0,temp0,N/2);
    negative ^= Subtract(v0,v1,temp1,N/2);

    Multiply(temp0,temp1,temp2,N/2,nv);

    if (negative)
      NegAccumulate(p1,temp2,N);
    else
      Accumulate(p1,temp2,N);

  } else  // v1=0: the terms involving v1 are not needed
  {
    const ushort* u0=U;    // pointers into the allocated space
    const ushort* u1=U+N/2;
    const ushort* v0=V;
    ushort* p0=P;
    ushort* p1=P+N/2;
    ushort* temp0=P+2*N;
// Optimizes the case of asymmetrical problems (V much smaller than
U).
// The first or first few recursions may need only 2 multiplications, but
deeper levels
// may not be asymmetrical and the main scheme (above) will kick in
then.
    Multiply(u0,v0,temp0,N/2,nv);
    Accumulate(p0,temp0,N);

    Multiply(u1,v0,temp0,N/2,nv);
    Accumulate(p1,temp0,N);
  }
}

ulong RoundUp(ulong N)
// Rounds N up to a value X * 2**Y, where X <= kMultiplyThreshold.
// This ensures that terms can be split into equal halves recursively
// until there are <= kMultiplyThreshold terms.
{
  ulong mask=-1,m2;
  while ((m2=mask>>1)>N) mask=m2;
  mask -= mask>>kMTbits;
  if ((N & mask) < N)
    N = (N & mask) + (mask & (mask>>(kMTbits-1)));
  return N;
}
```

BigMult.h

```
typedef unsigned short ushort;
typedef unsigned long ulong;

// Constants and prototypes for the faster multiplier.
enum {
  kMTbits       = 4,
  kMultiplyThreshold   = (1<<kMTbits)-1
};
void  Multiply(const ushort* U,const ushort*
V,ushort* P,const int N,const long nv);
ulong RoundUp(ulong N);
```

By William Porter

Lasso Studio for Dreamweaver 1.5

Web database programming for the rest of us?

Introduction

Lasso is a tag-based programming (or, if you prefer, scripting) language used to send instructions to the Lasso Web Data Engine (WDE). The Lasso WDE runs on a web server as a plug-in or CGI, where it is typically used as middleware to enable the web server software to talk to a back-end database such as 4D or FileMaker Pro. While the Lasso language itself is fairly high-level, it is nevertheless a reasonably complex language with hundreds of tags, and the fact that Lasso code gets scattered around in the HTML code of web pages makes writing it and reading it somewhat difficult. Until recently, Lasso developers had no choice but to rely entirely upon text-editors like BBEdit to do their coding. But with the release of the Lasso Studio for Dreamweaver (LS/DW) in December 1999, Lasso developers finally got a graphical interface to work with. And with the version 1.5 update, released in early Spring 2000, the LS/DW is ready for prime time. Lasso hasn't gotten any easier and doing advanced work with Lasso still requires that you get your hands dirty. But building a basic Lasso-powered site is now truly within the reach of non-programmers.

The LS/DW package from Blue World replaces what used to be known as the Lasso Developers Edition. The former Developers Edition was about testing and nothing else. It gave the developer a restricted-use copy of the Lasso Web Data Engine (WDE), so you could web-enable and test your pages on your development machine or demo them for your client. The Lasso Studio for Dreamweaver provides this as well, but it adds a set of extensions for Macromedia's powerful web site design program, Dreamweaver, so you can build Lasso pages right in Dreamweaver, using the Lasso wizards and tag objects as supplements to create pages quickly and with a minimum of direct involvement with the underlying code. What if you don't use Dreamweaver? Then part of the Lasso Studio for Dreamweaver will be a waste of money for you, and you cannot buy the web-enabling part of the Lasso Studio without the Dreamweaver extensions. (Adobe recently announced that GoLive 5 (their web-page editor) will be extensible in the way that Dreamweaver is already. Since the Lasso Studio for Dreamweaver extensions are written in JavaScript, it does not seem unrealistic to hope that Blue World will provide a Lasso Studio for GoLive in the future as well. But Blue World has announced no such plans.)

Like Lasso itself, the Lasso Studio for Dreamweaver does not care much what DBMS it interacts with. Lasso WDE can communicate with 4D, FileMaker Pro and ODBC data sources. For the purpose of this article, we are assuming that the database being published is running under 4D Server.

The Configuration Wizard

Let's imagine that you have kept your used bookstore's inventory in 4D for years and now you want

William Porter, Ph.D., is the owner of Polytrope Solutions, a database development firm in Houston, Texas, specializing in 4D, FileMaker Pro and Lasso projects. You can write him at wporter@polytrope.com.

to put the inventory online using the Lasso Studio for Dreamweaver. In order to get started, open the database under 4D Server, crank up the Lasso Web Server and finally, launch Dreamweaver. If everything is properly installed and configured, you'll see a number of new items in the Dreamweaver Commands menu (see **Figure 1**).

Figure 1.

If you are working on your site for the first time, the Lasso Studio will need to know some things about your database files: their names, the names of the fields, layouts and value-lists in each file. Gathering this information is the job of the Configuration Wizard, which uses Lasso to query the databases and store this information in a "snapshot" file.

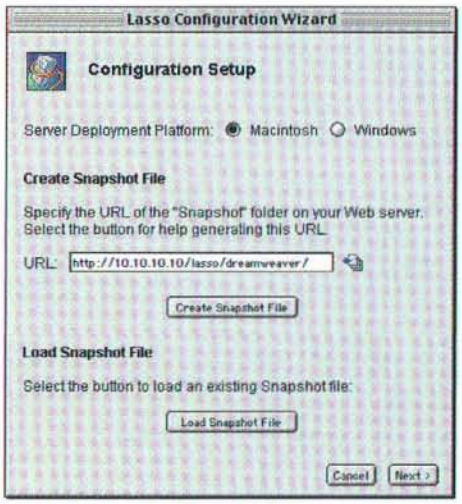


Figure 2.

The snapshot file is the key to everything. Without it, you can't work. With it, you don't even have to have 4D or the Lasso Web Server open to create Lasso pages in

Dreamweaver. This makes it possible for teams of developers to share snapshot files without actually sharing the databases or needing to have Lasso running on every machine. And if you work on multiple projects, you can create multiple snapshot files, name them, then load the one you need for your current work session.

Considering the importance of the snapshot file, it's unfortunate that the Configuration Wizard, which creates it, is the weakest part of the Lasso Studio. If you want to create different snapshot files for different projects, you'll have no choice but to do that manually, because the Configuration Wizard does not ask you to name the file it creates. More seriously, some users have found it difficult or impossible to use the Wizard with success, although the latest release of the Lasso Studio for Dreamweaver extensions fixed all the problems I myself experienced earlier. It is important to keep in mind that the LS/DW configuration wizard will only work if you have Lasso installed properly installed and configured properly and if your database is properly registered with Lasso security. These are tasks that beginning users — LS/DW's audience — sometimes find confusing. Fortunately, as a last resort, it's possible to create a snapshot "manually," if you have to.

When the configuration wizard is done, it will display the names of all the data sources it found (the names of your open files) and ask you to select one to work with. You can change your selection at any time later on by using the Database Selector dialog (**Figure 3**).

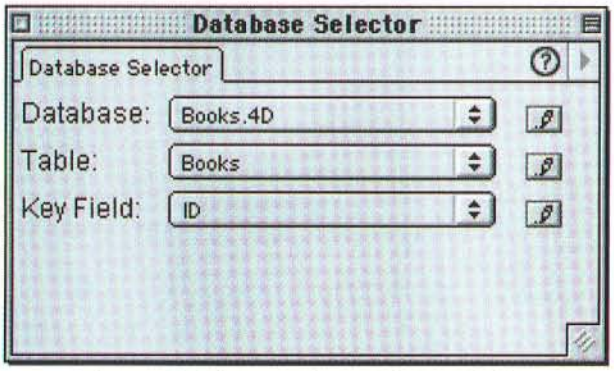


Figure 3. *The Database Selector dialog.*

Lasso Studio Basics

The Lasso Studio for Dreamweaver adds four pages to the Dreamweaver objects palette. These contain icons that you can use to insert tags that access data sources, tags that create Lasso forms, tags that rely upon Lasso extensions, and programming tags (like conditions, tokens, file manipulation tags, and others).

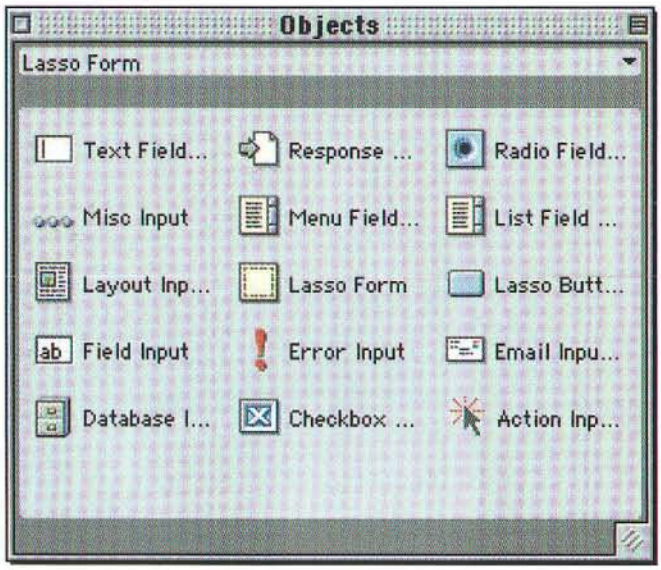


Figure 4. *The Lasso Form objects.*

For example, you can create a one-page Lasso search form using the Lasso Form page of the palette (**Figure 4**). With a blank document open, click on the "Lasso Form" object to insert a form area. (For some reason, the Lasso Studio will place the form at the top of the editable area regardless of where the insertion point is. You can cut and paste the form where you want it to be if necessary.) When a new form is created, the Lasso Studio automatically inserts a few objects into the form for you: objects that let you define the database to be searched, the layout to be used, and an object that identifies the web page that will display the resulting hit list. When you build a page "manually" this way, you must click on each of those objects and supply a parameter; that is, select the response page object and in the Dreamweaver object properties inspector, enter the name of the .lasso page to be used as your format file (**Figure 5**).

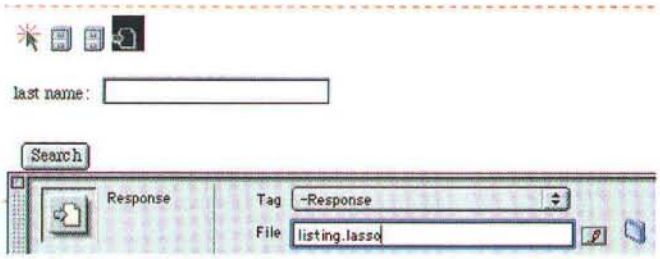


Figure 5. *The response page object shown above has been selected inside the Lasso form, and the parameter required by that tag (the name of the response or format file) is entered in the Dreamweaver object properties inspector.*

The icon of the clicking arrow shown as the first tag in **Figure 5** is the *action* tag, which specifies the action to be performed when the user clicks on the submit button. For a search form, that action would be "-search." Like the other Lasso objects in this row on the page, the action object represents a hidden input tag in the HTML. Remember, GUI and WYSIWYG are not synonyms! The Lasso Studio for Dreamweaver provides a graphical-user interface for inputting programming tags, but to an even greater extent than Dreamweaver's own page-editing interface, the Lasso Studio GUI is *not* WYSIWYG. With any luck, when your site is deployed, what you'll get on most of your Lasso pages is data, and that is one thing you *never* see when you're working in Dreamweaver.

Anyway, once you have all the hidden input tags the form needs, you just throw in a few search fields (using the same page of the objects palette) and add a submit button at the bottom of the page and you're done—without writing a single line of Lasso code and without ever having to open up Dreamweaver's HTML editor!

The Site Builder

But wait, it gets better. Why build individual pages one tag at a time, when the Lasso Studio can automatically generate all the pages for your entire site? Well, maybe not your *entire* site, but the Site Builder does build *sets* of related pages.

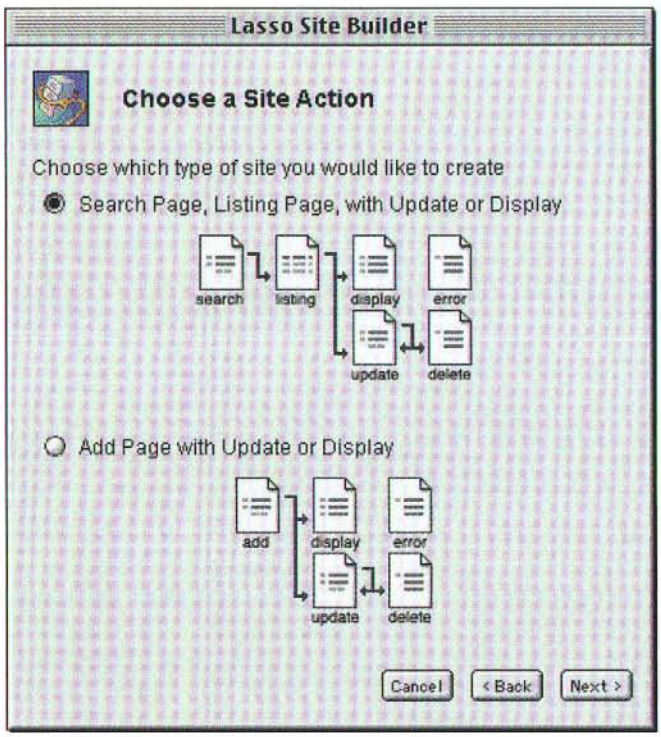


Figure 6. *The Site Builder's two page-set options.*

You start by selecting a "site action" in the Site Builder dialog (**Figure 6**), and then the Site Builder asks you a series of questions: what fields you want on each page; what field to use as the link-to-detail field on the hitlist page; whether to include a "find all" button on the search page; and so on (**Figure 7**).

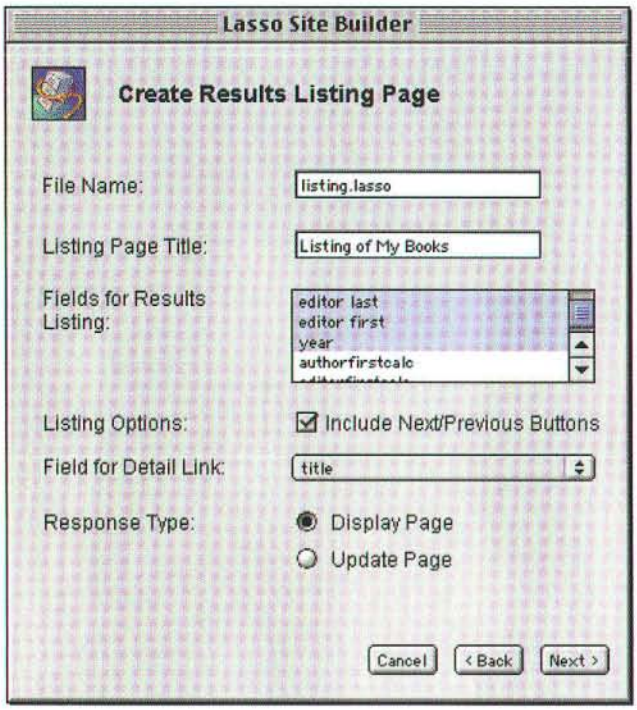


Figure 7. *The Lasso Site Builder walks you through the choices you need to make before it can build the page set for you. This screen shot shows the options available for the hitlist page, including (at the bottom) whether the linked detail page should be a display or an update page.*

When you get to the end of this interview process, the Lasso Studio performs what many older Lasso users will regard as a nothing less than a minor miracle: it creates an entire set of ready-to-run pages for you, in less than a minute. Once again, you did not have to write a single line of code — and this time you did not even have to type parameters in the object properties inspector. The Lasso Studio for Dreamweaver Site Builder is the star attraction in the package and beginning users will love the Lasso Studio for this feature more than anything else.

Lasso Studio for Dreamweaver 1.5 still has the Form Builder that was introduced in version 1.0. The Form Builder creates single pages and is particularly useful for editing the page sets generated by Site Builder. You can, for example, use the Form Builder to add a new page, then simply tweak the sequence of links in the revised page set.

Together, the Site Builder and the Form Builder allow developers to create a working draft of a basic site very quickly. Of course, the Lasso Studio does not make your pages look good for you., but it does not get in your way, either. When you use the Site Builder, you can tell it to use a Dreamweaver template and what editable region of that template to insert the Lasso code into, so elements like site headers and common menus can be placed directly into the Lasso pages by the Site Builder.

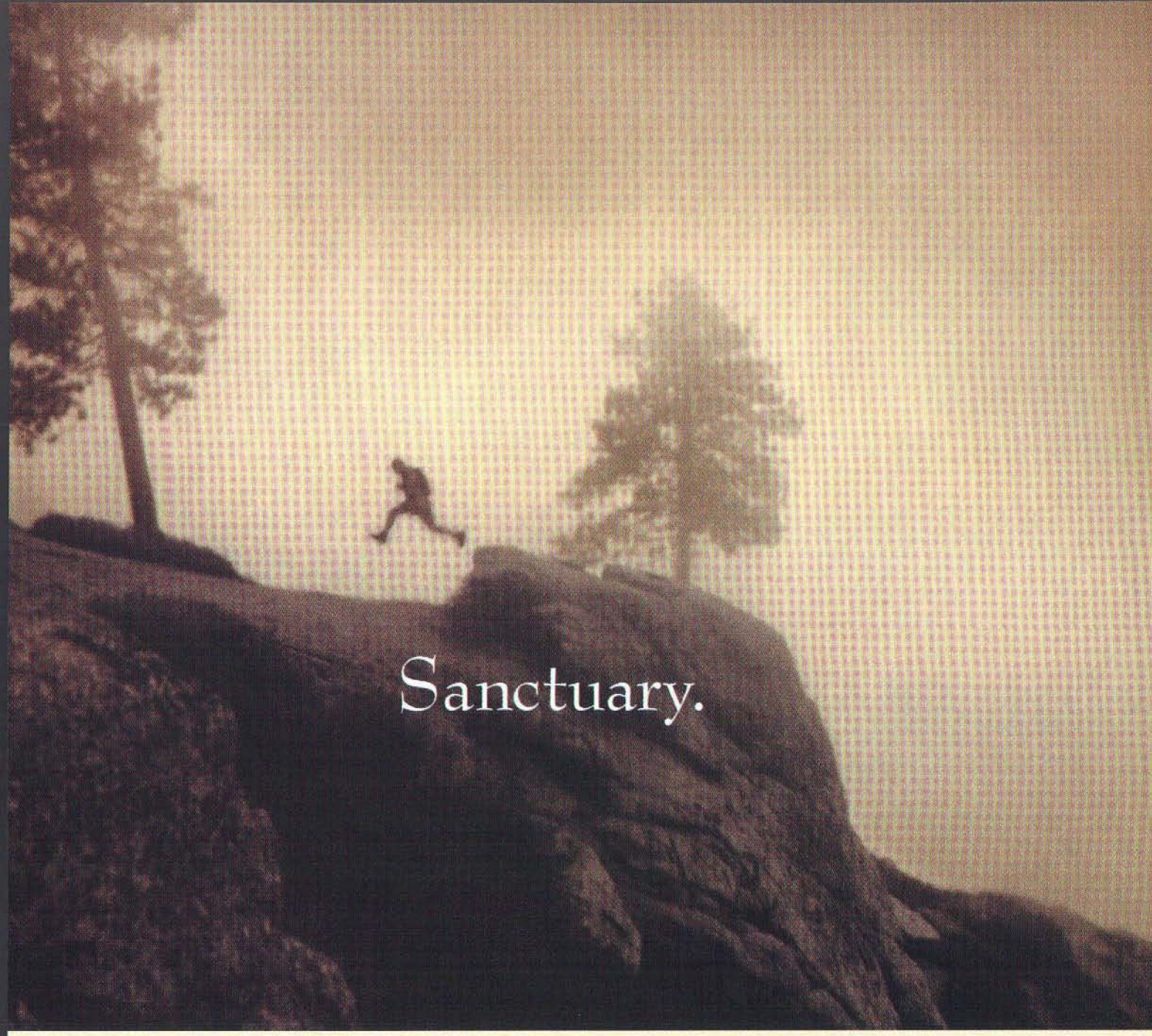
Beyond the Basics

Yes, there is a catch. The Site Builder does what it does and that's all that it does. For everything else, you're on your own with the Lasso Reference Manual. Working with Lasso remains a form of programming, and especially if you want to go beyond the basics, you will need to understand Lasso's programming logic, because the GUI requires that you know not just what tags (objects) to insert, but also what parameters to add to tags, and in what order to insert tags. And ironically, when you know enough to do it using the Lasso Studio's GUI, you know enough to do it without, so the advantages of the GUI seem less clear. Advanced developers may even feel that using the GUI is more trouble than working with the code directly.

One of the places where this limitation is most evident is in the way the Lasso Studio handles inline tags. Inline tags are placed within the format file that displays their results and they are processed before the page is displayed. For this reason, they are never seen by end users, either in the browser's location field or in the source code window. So, for example, you might create a page named "list_everything.lasso" which includes this inline:

```
[inline: datasource='Books.4D', table='books',
keyfield='ID', sortfield='author last', findall]

  [records]
    [field:'author_last'], [field:'author_first']:
<i>[field:'title']</i>
  [/records]

[/inline]
```

This simple code will find all the records in the database "Books.4D," sort them by the author's name, and display a hit list showing two fields, "author" and "title." Best of all, no information about your databases will be revealed in the source code for the page. Most advanced Lasso developers prefer to use inlines as much as possible. Yet Blue World does not promote their use energetically. The Lasso Studio wizards create page sets exclusively using the traditional tag types. If you prefer to use inlines, then the Lasso Studio for Dreamweaver's strongest feature — the Site Builder —



Move to digital.forest. The leading Internet Service Provider for the Macintosh community.

You need maximum flexibility, security and performance from your ISP. You'll find it with digital.forest. We're the proven provider of Macintosh Internet service.

Founded in 1994, our specialty is Mac systems and technologies. In fact, digital.forest is the world's largest Macintosh Hosting Provider. And our hard-earned knowledge of Mac server software is the broadest in the business.

We pioneered Macintosh Server Colocation and FileMaker Pro database hosting services. And our commitment to service keeps growing. So whether you choose us to host your files or house your servers, you've chosen the best.

Go with the industry leader. Discover the sanctuary of digital.for-

digital.forest

877-720-0483
info@forest.net www.forest.net

Call toll-free in the U.S. and Canada.
For international inquiries, please call 425-483-0483.

FileMaker Solutions Alliance Associate

will be wasted on you. And creating an inline in the Lasso Studio is, in my opinion, more difficult than it is to type the same code into the HTML editor by hand. Here is the inline inspector used to create the same tag that was shown on the previous page:

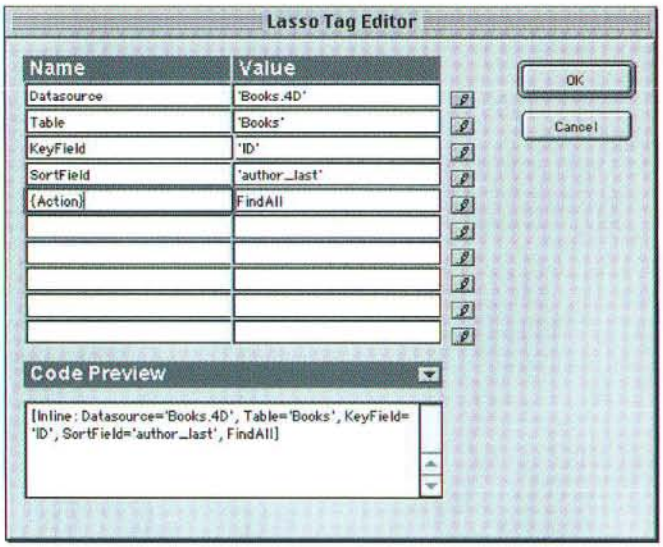


Figure 8. *The Lasso Tag Editor, being used to create an inline tag.*

The down side to inlines is that they are somewhat more complicated conceptually than traditional tags. Processing an add or search form using an inline in the response page rather than hidden input tags in the form page itself requires careful use of the [form_param] tag at a minimum, and in some cases can require the use of tokens to pass data from one page to the next.

In some cases, you simply cannot use the Lasso objects to insert code and must type it in manually. For example, it is common to use form parameters or tokens or cookies in links that conduct searches:

```
<a href="action.lasso?-datasource=Books.4D&-
response=viewcart.lasso&-
table=books&customer_ID=[token_value:'customer_id
']&-search]">Click here to view the items in
your shopping cart</a>
```

That link tells Lasso to go to the layout "web" in the database "items" and find all the items that have been selected for purchase by this customer, based on the value of the token "customer_id." Now there is a Lasso token object in the objects palette. But you cannot insert an object—Lasso object or other kind of object—into the object inspector's link field. And since the link field is so small, you will almost certainly want to type that tag into your HTML editor instead.

Availability and Pricing

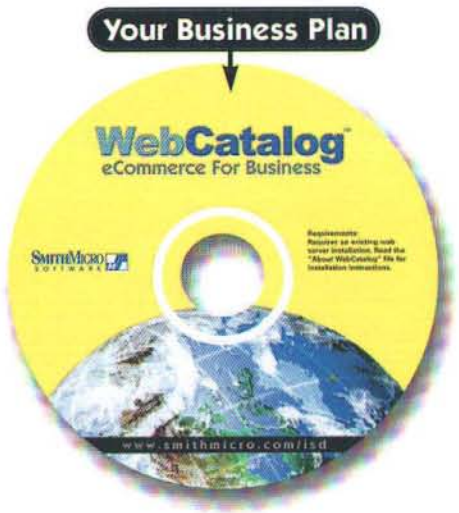
The Lasso Studio for Dreamweaver provides you, the developer, with two things: (1) a testing-only version of the Lasso Web Data Engine, with all the data-source modules (4D, FileMaker Pro and ODBC), and (2) the extensions for Dreamweaver. You must supply the databases and your own copy of Dreamweaver 3.01. A thirty-day evaluation version of the Lasso Studio for Dreamweaver can be downloaded from Blue World's web site: http://www.blueworld.com/download/. The Lasso Studio for Dreamweaver can be purchased directly from Blue World Communications. As of mid-summer 2000, the boxed version (with CDs and printed manuals) has a list price of $349; the electronic (downloaded) version is priced at $299.

Conclusion

On balance, version 1.5 of the Lasso Studio for Dreamweaver is a winner. The Lasso Site Builder is a godsend for beginners and a time-saver even for experienced developers. As we have seen, its difficult or impossible to do everything using the Dreamweaver GUI, and Hardhard-boiled Lasso coders may will not want to give up their favorite text editor, . but But I thought of myself as at least a medium-boiled coder, and I have learned that I can use BBEdit and the Lasso Studio for Dreamweaver side-by-side to very good effect. MT

Put eCommerce to work for you and change your business model.

WebCatalog is a complete eCommerce and dynamic web publishing solution. Utilizing existing web server software, WebCatalog provides the capability to design and operate online storefronts with all the features found in leading sites. Build unlimited storefronts with our Storebuilder wizard directly from your browser - even upload your product graphics! WebCatalog provides multiple platform support for Windows, Macintosh and UNIX. It's the easiest way to take advantage of the web and to change how...and where... you do business.



- **Unlimited storefronts**
- **High-speed internal database**
- **WAP-enabled sites**
- **ODBC, SQL support**
- **Electronic shopping cart**
- **Invoice calculations**

1.800.477.1543 • www.smithmicro.com/isd

SMITHMICRO SOFTWARE

The Legend of Arnold's Gold

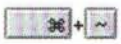
Ever since CodeWarrior DR/1 shipped back in late 1993, CodeWarriors have been asking for keystroke navigation features and other shortcuts for just about every part of the IDE. If you're like me, these manage to remain secret over time because it's time consuming to read the manuals every release to see what's changed. Some of these are not so secret but they are worth mentioning anyway.

With that in mind, here's the torrid exposé that dares to reveal them all.

Search Results and Error Message Windows

Cycle through list of found items when the keyboard focus is on the editor pane. For searches, the text that was found is highlighted as you navigate through the list.

Editor Window



With a source file, opens its corresponding header file or vice versa.

The button just above the vertical scroll bar is a pane splitter that allows you to view different parts of your file at the same time.

The icon just above the splitter toggles the toolbar position between the top and bottom of the window.

Double-click on a word in the editor selects it.

Triple-click on a word selects the whole line that contains the word.

Moves the insertion point to the beginning or end of the document.

Moves the insertion point to the top and bottom of the visible page. If you are already at the top/bottom, pressing this key combination again will take you to the next/previous position.

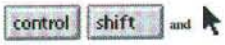
Select the text between the last and current insertion point. This can also be used to extend a text selection in either direction.

Selects all the text or extends the selection between the insertion point and destination. Follows similar logic as non-shift case.

Richard Alexander David Atwell, a.k.a. "Ratwell," is a Mac OS Debugger Engineer at Metrowerks and is responsible for making MetroNub do nasty things so you can debug your code. Good ideas for CodeWarrior t-shirts can be sent to ratwell@metrowerks.com.

Select/extend the text between the insertion point and destination. Follows similar logic as non-shift case.

Navigate sub-words by case. For example, if the insertion point is at "Debug" within "DebugText", the insertion point will move to the start of "Text".



Extend a text selection within using the above logic.

When you have browsing active for your project, option-click on a symbol name takes you to the definition of the symbol, if there is only one. If there are more than one, the Symbol Browser window opens up.

Same as above but searches using the reference application specified in the IDE Extras preferences panel as a backup.

Hierarchical Lists

Collapse/expand a node and all its sibling nodes. We call this "wide disclosure." You can also click the mouse instead of using the left and right arrow keys.

Collapse/expand a node and all it's sub-nodes. Again, you can also click the mouse instead of using the left and right arrow keys.

Combine the above behaviors and perform a deep and wide disclosure, sometimes called recursive.

Project Window

Typing while viewing the Files page takes you to the file with the nearest alphabetically matching item or group (the item must be visible, i.e. not be in a collapsed group). You can see what you are typing at the bottom of the window.

Typing while viewing the Link Order page takes you to the file with the nearest alphabetically matching name.

Thereafter, pressing the tab key takes you to the next item that matches the text you typed.

You can drag files or projects from Finder onto the Project Window to add them to a project. If you have more than one target per project you will be asked which targets to add those files to and they will be filtered by the target's file mapping settings.

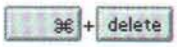
In the Files page, clicking on a header sorts using the items in that column. Clicking on the button to the far right of all the headers when a column is sorted toggles the sort direction much like the behavior of the Finder.

In the Files and Link Order pages tries to open that item in the editor.

In the Files page edits the name of the selected group.

In the Targets page opens the target preferences for a selected item.

In the Targets page, all project commands apply to the selected item, so you can use continuous and discontinuous selection to select which takes to build, for example.



In any of the project pages, presents the delete item dialog.

Compare Files Window

You can drag and drop files and folders from the Finder into the icon wells at the left of the window.

Breakpoints and Watchpoints Windows

Click on the red dot next to the name of the function/filename toggles the state of all breakpoints in the list.

For watchpoints or breakpoints an editor window will open that reveals the selected item.

List Views

You can use type ahead in the list views to select the item with the nearest matching name.

Use can use the arrow key to navigate within the list.

Variable and Register Views

Edit selected values.

Navigate within the list.

Bring up a contextual menu of applicable items from the Data menu.

Double-click on a variable will open it in its own window.

Expressions Window

Drag and drop variables to this window.

Drag names from the register window to this window.

Use ®R1 (option-r is the copyright character) to specify registers by name.

Stack Crawl Window

You can drag the blue arrow that represents the current program counter register forward or backward to change the next line to be executed. Be sure you know what you are doing when you attempt this, i.e., don't move the arrow into prologs or into other functions.

Clicking the small document icon at the top left of the editor pane opens the source file in an editor.

Clicking the small dot icon at the top left of the Variables pane toggles between all the variable display modes.

Any Window

In the close box of the window title bar of a document window closes all windows of that type. For example, all editor windows or all project windows or all variable windows, etc.

On the title bar of a window to move it without making it the front window after you release the mouse button.

Find Dialog

Cancel a find.

Drag files from the Project Window onto the files list to add them to the search.

Drag a folder from the Finder onto the files list to add the contents to the search to find recursively (use with caution as the list can take a long time to build with large directories).

Toolbars

Remove items from the toolbar.

Mouse over a button will display the balloon help for that item.

Credits

Thanks to Max and everyone else at Metrowerks who helped to collect the information for this article. If you'd like to get in touch with us about CodeWarrior issues, post in our newsgroup or email us directly.

- Newsgroup: comp.sys.mac.programmer.codewarrior
- Technical Support: cw_support@metrowerks.com
- Report Bugs: cw_bug@metrowerks.com
- Suggestions: cw_suggestion@metrowerks.com


AUSTIN
BOSTON
NEW YORK
CHICAGO
LOS ANGELES
WASHINGTON, DC
SAN FRANCISCO
Sandy Conroy. | PROGRAMMER
(Now she designs web sites, too.)
Your web. Your life. Your conference.
WEB design & development
www.mfweb.com
PLATINUM SPONSOR
SILVER SPONSORS
Adobe
interland
real
Media3
Nextera

By Andrew C. Stone

Installing Mac OS 9, OS X and X Server on a G3 Laptop

During these hybrid times, Macintosh developers are concurrently developing software for Mac OS 9, Mac OS X, Mac OS X Server and Darwin. I realized it would be optimal to allow my 1 year old and hopelessly out of date WallStreet G3 laptop to boot off of any of these systems, without dragging around an external hard disk. It's possible to do this, but you have to be careful about the order of installation and the order and size of partitions. With some tips from Joe Keenan from Apple, I was able to accomplish the install without too much trouble. This article will go over the process step by step, with tips and tricks along the way.

Step 1. Make a partitioning plan for your internal hard disk.

The most important step is for you to determine how you want to carve up your disk. If you consider the fact that new versions of OS X are coming out fairly regularly, then the partition devoted to OS X should not contain any vital source or "permanent" files. Since Mac OS 9 is stable, you might consider that partition as the place to keep important files and data. You'll want at least 3 HFS extended partitions.

If your disk is larger than 8 gigabytes and your laptop has the "Old world ROM" (WallStreet and earlier models), you'll need to squeeze the operating systems onto the first 8 gigs. Old World machines did not implement the necessary standard for doing ATA block reads past 8GB, so all boot partitions must begin before 8GB. You might consider making a large forth or fifth partition of the remaining disk space if you have a large capacity hard drive.

Because the newer boot loader that comes with X can boot either OS X or OS X Server, while the Server boot loader cannot load X, you have to install X on a partition before Server. Since you need a Mac OS 9 partition to run various USB devices at the time of this writing and the boot chooser "System Disk" runs as a Mac OS 9 System Extension, 9 needs to be installed on the first partition. This will allow you to boot with the option key down and select one of the various operating systems from which to boot.

So, on an 8 gig drive, you might divide it up into 3 equal partitions of about 2.6 gigs each, and install Mac OS 9 on the first partition, Mac OS X on the second, and Mac OS X Server 1.2 on the third. Because Darwin 1.0 is based on the exact same code as OS X, you'll be able to build and tinker with Darwin on the OS X partition beginning with Developer Preview 4.

NOTE - if you have trouble with any of these steps, you might try hooking a SCSI disk to your laptop. I noticed that booting off of the CDs sometimes hung, but when the external SCSI disk was connected, it had no trouble booting.

Be sure you have the following Apple CDs, or later versions where applicable:

- **Mac OS 9**
- **Mac OS X Install CD**
- **Mac OS X Server 1.2**
- **DeveloperCD**

Andrew Stone <andrew@stone.com> is the chief executive haquer at Stone Design Corp <http://www.stone.com/> and divides his time between raising children, llamas & cane and writing applications for Mac OS X and playing with Darwin.

Step 2. Partition the hard disk and Install Mac OS 9

Warning! Following these instructions will completely wipe clean your PowerBook! Be sure to back up everything to an external disk or network.

A. Insert the **Mac OS 9** CD. Premium and Select Developers received OS 9 in October 1999's monthly mailing.

B. Boot off the CD by holding down the C key while booting. If that doesn't work, try holding down the option key! If that fails, reset the Parameter Ram by holding down Option-Command-P-R while rebooting. You'll hear a second start up bong which indicates that the old settings have been reset to defaults.

If you plan to install Linux as a fourth operating system, you'll need to learn the command line tool "pdisk", which lets you partition the disk and assign non-Apple types to each partition. Moreover, you can erase certain partitions without destroying the data on the others. Otherwise, you can get by using the GUI application "Drive Setup" provided by Apple in Utilities/Drive Setup *f* on Mac OS 9.

C. Partition your hard disk
a. Start the Drive Setup application to partition your disk according to your plan.
b. Select the Internal Drive (ATA 0 0 0)
c. Click "Initialize"
d. Click "Customize"
e. Choose the number of partitions you desire, such as "3 Partitions" from the pop up button
f. Click in the area which represents the 3 partitions and resize each to your desired plan
g. Choose HFS Extended (HFS+) as the file system type for each of these partitions
h. Click "Initialize" and "OK"

D. Double-click "Mac OS Install" and click away until you can select the partition to install Mac OS 9 on - choose the first partition and click install.

E. Reboot and remove **Mac OS 9** CD.

Step 3. Install Mac OS X onto the second partition

A. Insert **Mac OS X Install CD**

B. In order to switch between operating system, you'll need to install the System Disk Utilities.
Copy the application **System Disk** in Mac OS X Install CD/Utilities/System Disk Utilities/ to the recently installed Mac OS 9 disk.

C. Double-click *Install Mac OS X* application

a. This will put up a panel that explains the reboot into OS X - click "Continue"
b. After rebooting, the installer gets loaded, click Continue, Agree, OK...
c. Choose the second partition and click all the OK's
d. OS X will get installed, and then will auto reboot

D. The Assistant's *Set Up* module automatically appears so you can configure your computer

Just like in OS X Server, you can configure the following:

- keyboard, root password, Web, Quicktime Streaming and Apple File servers
- ethernet ports, hostname, router, inet mask, IP address,use NetInfo
- remote login, time zone, time, server configurations, accounts, auto login

You can always come back to these via /System/Administration/Assistant.app -> Set Up Assistant

Click restart to test your network connectivity and have these changes take effect.

E. Install user applications. Check www.omnigroup.com, www.stone.com, etc. for the latest!

G. Remove **Mac OS X Install CD**

Step 4. Install Mac OS X Server 1.2 onto the third partition

A. Insert **Mac OS X Server 1.2** CD

B. Boot off of the CD or Mac OS 9 partition

C. Double-click the *Install Mac OS X Server* application
a. Choose your installation language: English, French, German, Japanese.
b. You'll get a panel - "Mac OS X Server cannot be installed on this computer"
c. Choose Special->Configurations
d. Click the check box "Allow installation on unsupported configurations" and "OK".
e. Click a bunch of Ok's and "I Agree".
f. Select the third partition, and OK, OK.
g. The install continues and then auto reboots

D. The Assistant's *Set Up* module automatically appears so you can configure your computer

Just like in OS X Server, you can configure the following:

- keyboard, root password, Web, Quicktime Streaming and Apple File servers
- ethernet ports, hostname, router, inet mask, IP address,use NetInfo
- remote login, time zone, time, server configurations, accounts, auto login

You can always come back to these via /System/Administration/Assistant.app -> Set Up Assistant

Click restart to test your network connectivity and have these changes take effect.

E. Remove the **Mac OS X Server 1.2** CD as the computer reboots.

F. Install the Developer software
a. login as 'root'
b. Insert the **DeveloperCD**
c. Install WebObjects and Development Tools: launch Installer.app by double-clicking WebObjectsDeveloper.mpkg
d. Remove the **DeveloperCD**
e. Insert the **Mac OS X Server 1.2** CD
f. This is important - WebObjects 4.0 "writes over" some upgraded files in Server 1.2, therefore, you'll need to install this on top of the Developer install. Double-click the 'UpdateWO4.0_Developer.mpkg' in /Mac_OS_X_Server_1.2/System/Installation/Packages/
g. Click Install
h. Remove the **Mac OS X Server 1.2** CD

G. Install user applications. Check www.omnigroup.com, www.stone.com, www.stepwise.com, etc. for the latest! A suite of web design tools is available at: ftp://ftp.cs.unm.edu/pub/stone/MacOSX/StoneWebTools.1.4.m.P.b.tar.gz

Conclusion

Being a freewheeling and footloose cross-platform developer is a tricky but obtainable goal. Turning your G3 laptop into a multi-OS boot machine is a worthwhile endeavor for mobile development and remote demos, and it really impresses the Windoze users in the airplane seats next to you! MT

PRODUCTIVITY ENHANCER!

MacTech Magazine is where people look to find detailed and in-depth information on Macintosh technology and development. Each issue covers the latest and most detailed information on what makes the Macintosh number one!

If just one article, in just one issue, solves even one problem for you, the subscription has paid for itself!

Order Online today at www.mactech.com

P.O. Box 5200, Westlake Village, CA 91359-5200
Voice 800/MACDEV-1 • Fax 805/494-9798

by Jeff Clites <online@mactech.com>

Quartz and PDF

As you've probably heard, Apple's new graphics system, Quartz, is based on PDF. You've seen screen shots of Quartz, and undoubtedly you've encountered PDF documents before, but that's probably where your knowledge of PDF ends. While you really don't need to know anything about PDF in order to develop applications for Mac OS X, it always helps to know a little bit more about the technologies you are working with, so that you can fully leverage them in developing your products. This month we are going to take a closer look at Adobe's PDF and its role in Apple's new imaging system.

So what is PDF exactly? The acronym stands for "Portable Document Format", as you probably know, and it is fundamentally a file format. In fact, Adobe's description, stated at the beginning of their PDF Reference Manual is, "PDF is a file format used to represent a document in a manner independent of the application software, hardware, and operating system used to create it. A PDF file contains a PDF document and other supporting data." This is in stark contrast to PostScript, which is a full-fledged programming language. So at first blush it may seem a bit strange to base a graphics and windowing system on a file format—Apple refers to PDF as a superset of PostScript, but now this seems somewhat like calling Photoshop a superset of C++. So what's the deal? Things begin to make a little more sense when you learn that PDF (and specifically, a PDF file) has four conceptual parts: a set of basic data types, a file structure, a document structure, and a page description. It is this last part, the page description, which is the core of PDF-as-a-graphics-model.

The Adobe Imaging Model

PostScript and PDF are based on the same conceptual imaging model, sometimes called the Adobe imaging model. At the most basic level, this model is no different from that of QuickDraw or most other 2D graphics systems: you create an image using operators which move a virtual pen to place marks on a canvas. In PostScript, you have primitive marking operators and in addition you have programming constructs such as loops and variables. In PDF, you retain these basic marking operators (and add others), but without the additional programmatic infrastructure, so a PDF document ends up being a description rather than a program. As a simple example, a page with 30 circles on it might be created in PostScript by placing the circle-drawing instructions in a loop which iterates 30 times; in a PDF file, this would be represented by 30 separate circle descriptions. As a consequence, a PDF document is potentially larger than a PostScript document, but it doesn't require the overhead of an interpreter to draw a page, it can render faster because the primitive marking operators are implemented directly in machine code, and it is easier to manipulate because strings and other components can be reliably located within a document (in PostScript you really have to "execute" the entire page first). You can think of this aspect of PDF as a RISC version of PostScript.

So, basing Quartz on the Adobe imaging model really means basing its primitive marking operators on those of PDF (and consequently of PostScript). These operators live in the part of Quartz called Core Graphics Rendering. (I'll call this "CG" for short, as Apple seems to have not officially named their new graphics model. I'll avoid just calling it "Quartz", as this includes Core Graphics Services, which is essentially the window server.) As I mentioned above, there isn't a huge conceptual leap from QuickDraw, but that doesn't mean that there aren't huge advantages. For one, distances in CG are given as floating-point numbers rather then integers, and can be in real-world units such as inches and centimeters. This difference is profound, as it largely divorces the drawing process from explicit concerns of screen or printer page size or resolution, and reduces such things to primarily a scaling issue. It's also fundamentally vector-based, so you're not tempted to draw a single "pixel", although even in QuickDraw one tends to do this by drawing a 1 x 1 square. Also importantly, CG has a sophisticated set of text layout and graphics transformation


• Custom Destinations
• Electronic Transaction Processing
• Easy Trialware Creation
• Install or UnInstall
Get It Together With InstallerMaker™
• Installation From FTP or HTTP Sites
When Time Is Money, Get It Done Fast!
Build Smaller Installers – With StuffIt InstallerMaker, you'll build installers faster than ever before! Compress your installers an average of 15% smaller using the power of the StuffIt Engine®. Smaller installers download faster, provide added space on CDs and servers, and increase network bandwidth.
Quickly Create Demoware - Easily turn your application into a polished demo. Just set the number of days for the demo to be active, paste in the graphics, and you're done!
Archive Freshening - Automatically update your installer project file; eliminate repeated searches for modified files.
• Resource Installation
• Built-In Resource Compression
Scripting Saves Time - InstallerMaker provides full scriptability for all features. Automating the build process saves time and helps ensure the integrity of your installer.
• ShrinkWrap Disk Image Support
• Marketing Opportunities To Help You Make Money
Trialware in Minutes - Creating trialware is a breeze with InstallerMaker. With just a few clicks, and no extra coding, you can create trialware that is e-commerce ready.
Update in a Flash - Easily build intelligent "diff" files in installers to create small updaters for quick online distribution. From one simple installer, you can update 68k, PowerPC or FAT applications.
• Hierarchical Package Support
More Details Online - There's a lot more InstallerMaker can do for you. Get all the info at www.aladdinsys.com.
• Built-In Updaters
Aladdin Systems
© 2000 Aladdin Systems, Inc. StuffIt InstallerMaker is a trademark of Aladdin Systems, Inc. Other products are trademarks of their respective holders.

operators, so finally it will be easy for a Macintosh programmer to rotate text and images on the screen. And on top of PDF's core, Apple has added features such as composition and translucency. It wouldn't be surprising if these were to migrate back into a future version of the PDF specification. (Also note that CG is based on PDF 1.2, rather than the current version 1.3, but Apple plans keep pace with the Adobe's specification as it evolves.)

Core Graphics Rendering at the Center

Core Graphics Rendering serves as the hub of drawing activity, accepting instructions using its native C-based API (which you'll likely access through a higher-level framework), QuickDraw commands, PDF files, or possibly other input sources. From here it can render them to the screen, create raster data for a printer, or record them in a format such as PDF or PostScript. (It's fairly trivial to record a sequence of CG drawing operations to a PDF document, much as a PICT could store a recording of QuickDraw commands. It's easy to forget that PICT is actually a vector format, isn't it?) Note that this *doesn't* mean that all graphics, before displaying to the screen, exist as actual PDF *documents*.

There are several key advantages to this arrangement. First, it brings the conceptual simplicity that Display PostScript brought to NeXT's operating systems, namely a single graphics model for both screen and printer. This makes things such as print preview and print-to-file trivial, since PDF is the default spool-file format. This also means that you can get perfect output from cheaper, non-PostScript laser printers, and it opens the door for a new generation of "dumb" printers with minimal computational power, because the operating system fundamentally understands how to render documents all the way to raster data at printer resolutions. (This is a tactic used by the first generation of laser printers produced by NeXT, which were cheaper and operated at a higher resolution than other printers available at the time, and which I'm excited to see finally make it to the Macintosh. It always seemed like a waste to pay for a slow processor which sits mostly idle in a printer when there is a much more powerful processor available in your computer.) Core Graphics has many technological improvements over Display PostScript, but it also has the very clever advantage of freeing Apple from Adobe's licensing fees and restrictions while maintaining a very similar API which allows long-time OpenStep developers to leverage their past experience.

Reading Assignment

In the coming months I hope to cover additional aspects of Mac OS X's new graphics environment. Until then, you have some reading to do. First and foremost, you should read Apple's *Inside Mac OS X: System Overview*. In fact, you should read the whole thing, from cover to cover, and then read it again, because it's all important information that you'll need to know as we move forward with Mac OS X. There are also several articles on the Ars Technica web site which augment this same material, including an article specifically on Quartz and Aqua. Second, take a look at Adobe's PDF specification — it's large but fairly readable. You won't have to know this information directly in order to use Core Graphics, but it helps to know a little bit about PDFs in greater depth, since you'll be working with a lot of them. This reference also spells out the connection between PDF and PostScript more fully. If you're interested, you might also do a little light reading on PostScript, mostly for perspective rather than for the details. Adobe's *PostScript Language Reference* has a great deal of explanatory material, and you can also download the electronic version of Glenn Reid's *Thinking in PostScript* book, which is much shorter. (Somewhat ironically, even the PostScript references come in PDF form.)

Inside Mac OS X: System Overview
<http://devworld.apple.com/techpubs/macosx/System/Documentation/Developer/SystemOverview/SystemOverview.pdf>
Ars Technica: Mac OS X Update: Quartz & Aqua
<http://www.arstechnica.com/reviews/1q00/macos-x-gui/macos-x-gui-1.html>
Portable Document Format Reference Manual Version 1.3
<http://partners.adobe.com/asn/developer/acrosdk/DOCS/pdfspec.pdf>
PostScript Language Reference, Third Edition
<http://partners.adobe.com/asn/developer/PDFS/TN/PLRM.pdf>
Thinking in PostScript
<http://www.rightbrain.com/pages/book-download.shtml>

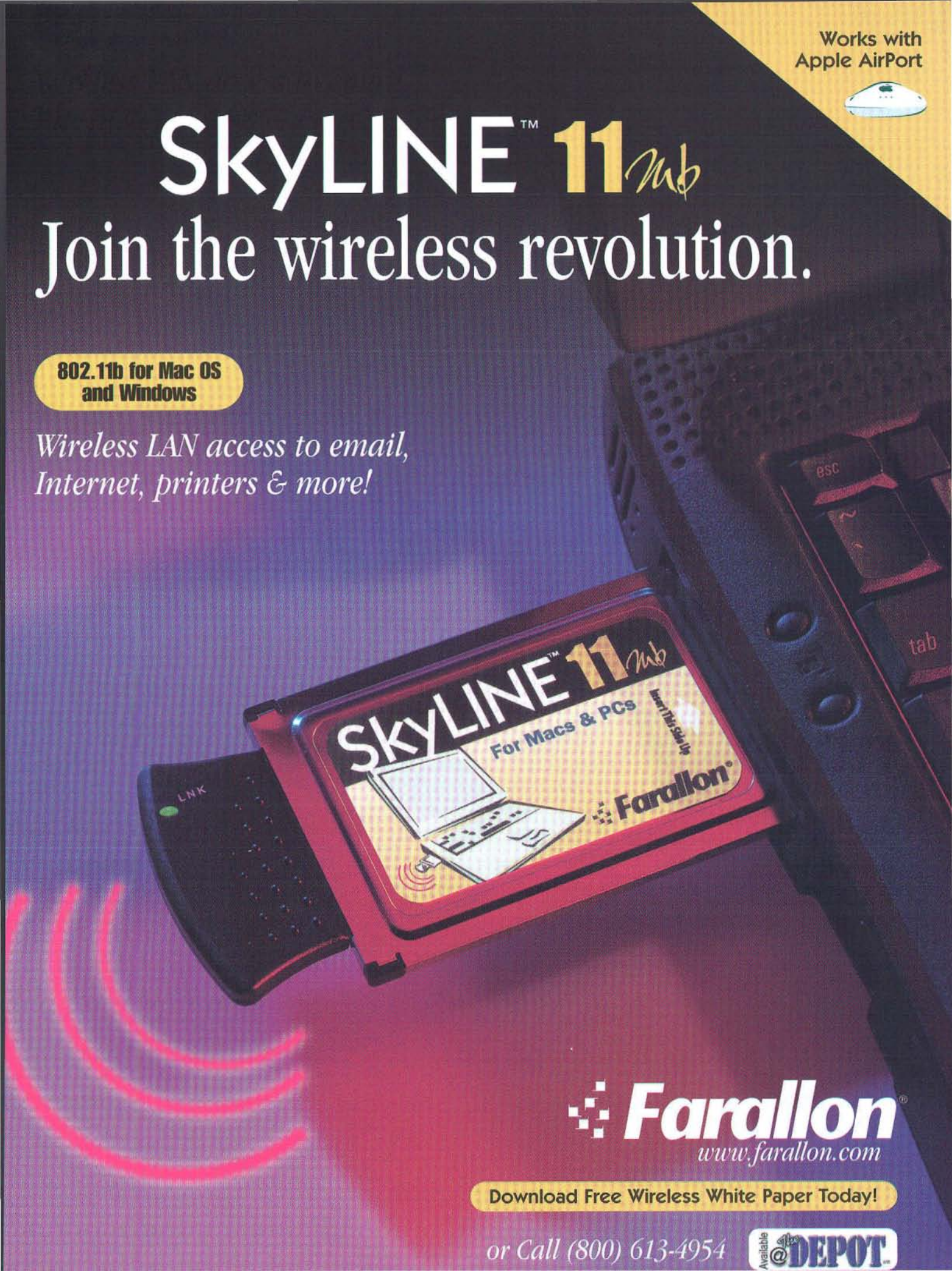
Works with
Apple AirPort

SkyLINE™ 11Mb

Join the wireless revolution.

802.11b for Mac OS and Windows

Wireless LAN access to email, Internet, printers & more!



Farallon®

www.farallon.com

Download Free Wireless White Paper Today!

or Call (800) 613-4954

Available @ the DEPOT.

By Sam Krishna and Patrick Taylor

The Not-So-Infinite Loop: Request-Response in WebObjects

In the beginning there was the Web...

When Tim Berners-Lee wrote Web.app (the first graphical web browser) using Nextstep tools, he couldn't have imagined the way the World Wide Web (1) would captivate an unsuspecting public. As revolutionary as it has become, Berners-Lee's creation wasn't a unique event. There were many predecessors to the Web which ranged from Vannevar Bush's ambitious, but ahead-of-the-technology, Memex to Ted Nelson's ambitious, but directionless, Xanadu. Unlike most of its predecessors though, the technology behind the Web was — and is — remarkably simple, especially when you consider how it is being applied today.

Most everyone knows at least a bit about how the Web works. With a browser that supports HTTP (hypertext transport protocol), a user connects to a remote server using an URL (Uniform Resource Locator) stored as either a bookmark or from a hypertext link on a document located at another remote location. The browser then proceeds to interpret the HTML tags in a text file on the remote server, formatting the text file based on the tagging. Hypertext links within index pages serve as springboards to other documents both within the current server or any number of servers around the world.

The number of those servers increased beyond anyone's reasonable expectations. By 1993, there were 50 web servers. Two years later there were 200 times as many. By 1998 there were well over 6 million and tens of thousands are added every day.

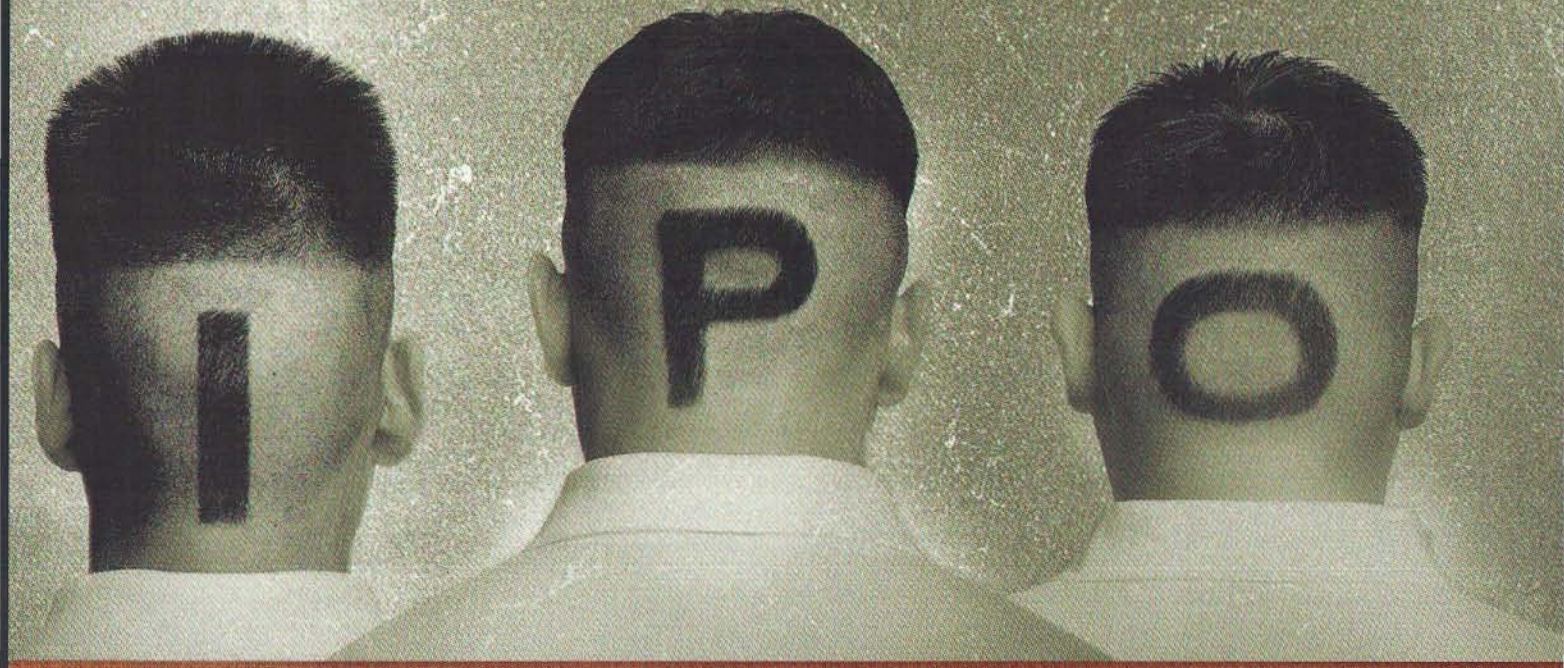
This was not the way the world was supposed to turn out. Back in the early 1990s, pundits were predicting that in the future information was going to be available to everyone on CD-ROMs with exciting multimedia interfaces: dictionaries, encyclopedias, games, cookbooks, repair manuals, baseball statistics; the knowledge of the world was going to be at your fingertips. As the speed of CD-ROMs leapt from 150 KBps to 300KBps and beyond, all manner of exotic data would be available ... for a reasonable price. It actually seems a bit funny looking back, but Bill Gates wasn't the only who didn't "get" the Internet until late 1995. Only the most visionary or lunatic individuals would have bet that the Web was the future much before then, though nowadays the whole world claims to have seen it coming as far back as 1991.

Of course, we know how things turned out. Tim Berners-Lee created the tool that provided the foundation for e-commerce, the rise of open source development as a mass movement and 11,000 point Dow Jones stock-markets. Would that all Nextstep apps have been so successful!

Good Enough, But Not By Itself

Berners-Lee wasn't out to revolutionize the world, but rather to create a useful tool that would let researchers at the European Center for Nuclear Research (CERN) share documents and data. Of course, it didn't take a nuclear physicist to see the advantages to hypertext linking and the simplicity of the Web's

BOOTCAMP FOR STARTUPS.℠
THE FEW, THE PROUD, THE OBSESSED.



LONDON, September 4–5 **BOSTON, September 20–21**

garage.com
we start up startups

Attention. Join Garage.com's two-day Bootcamp for Startups. Learn the fundamentals of taking your company from startup to IPO. Hear from the high tech industry's top investors, experts, and entrepreneurs. Gain invaluable information about raising capital, building a buzz, hiring top talent, and launching your product. At ease.

LOG ON TO **WWW.GARAGE.COM/BOOTCAMP** TO LEARN MORE & REGISTER TODAY.

Forbes CAPITALIST TOOL · SAFEGUARD · TESTA, HURWITZ & THIBEAULT, LLP · hp invent · Goldman Sachs · TechSpaceXchange · ADVANCED TECHNOLOGY · METRIUS · PricewaterhouseCoopers · quidnunc · Silicon Valley Bank · plural · StorageNetworks · THE STANDARD · IBM

Value versus Cost: the WebObjects Dilemma

I suppose this is as good a segue for what WebObjects costs as any. Unless you are fortunate enough to be an academic purchaser, a developer's licence for WebObjects/EOF will set you back US$1499. This isn't a licence to deploy only to develop, you will have to get a separate licence to deploy your application ranging from $5000 (for a 100 transactions per minute license) to $50,000 for an unlimited transaction, multiprocessor licence. Since every version of WebObjects includes the same software, all you need to do is enter a new licence key and the system will automatically accept a greater number of transactions (up to the limit allowed by hardware and the underlying operating system.

Academic developers can make out like bandits given Apple's incredibly generous academic bundle. For $99, academic (non-commercial) developers get a developers licence ($1499) and an unlimited transactions single-processor licence ($25,000). This may be the best reason yet to upgrade your university degree. MT

design over handling paper documents. But if all the Web had been good for was posting acronym-ladden documents and pictures of your cat, its growth might have stalled long ago.

Text and links gave way to a host of new datatypes. By adding new tags to a simple and flexible system, it was possible to extend the Web far beyond its early limitations: images, video, audio, animation and interactivity became realities as pioneering individuals pushed the envelope further and further and further.

However, while the types of documents changed dramatically, the way those documents were served didn't change quite so much. Apache today is a much better web server than HTTPd was in the early 1990s, but it still serves static documents in basically the same way. You can create amazing web sites, however if you build them with straight HTML pages the sites can't scale very well. It's a hassle to rewrite HTML and the hassle grows geometrically with the number of pages and their complexity. And if it's difficult to rewrite your own HTML, imagine picking up after someone else!

If you want to go beyond static web sites (coherent collections of text-based HTML documents) you need to move to scripting or web application servers. If you don't mind hand-coding virtually all of the dynamic behaviour in your site, you may opt to use Perl or any number of scripting languages to dynamize your website. Don't fool yourself, remarkably sophisticated sites have been created with scripting languages alone, but not everyone is prepared to start from scratch. Or has the patience or time to do so.

And while WebObjects isn't the only web application server, it is the most mature. WebObjects backed by EOF is an industrial strength foundation for demanding web sites, which may sound like the PR material you'll read about most web application servers, but in this case it's real. If your site tops off at

$ $ $ Buying online?

Will it work?

If you didn't know then, you will now. Introducing MacBuy.com. The first and only comprehensive buyer's guide from the editors of Macworld magazine; with MacBuy.com you'll know...

...which products to buy. **And which to avoid.**
The best deals on the Web. **And where to find them.**
How much it costs. **And how much it's worth.**

www.macbuy.com

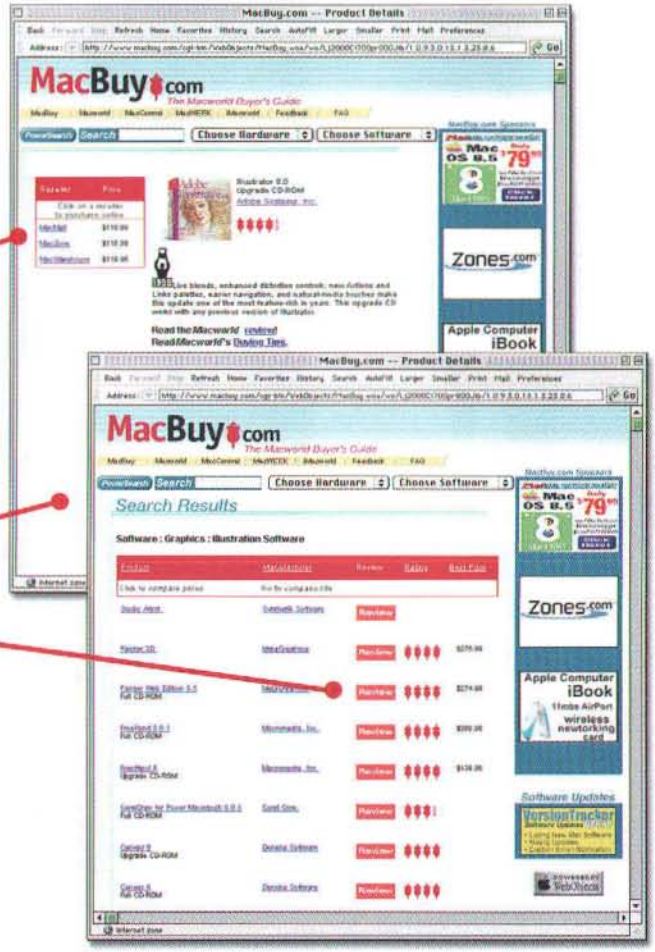
The Macworld Buyer's Guide

As Easy as 1-2-3
Whether you're searching for a product review or finding the Pick Of The Day, it's easy with MacBuy.com's clean user interface and intuitive navigation.

Convenience and Value!
The latest and lowest prices from leading Mac resellers. Plus, convenient links to their sites so you can purchase the product you want instantly! MacBuy.com tells you how to shop smart.

Powerful and Easy-...
to-use search engine. You can search by product or vendor name, or use the Power Search button to search by multiple criteria, such as rating and price.

Support from Apple
Literally. Built with Apple's WebObjects, MacBuy.com is solid as...well, a rock.



From the publishers of Macworld magazine

MacBuy.com

Get it right.

The WebObjects Team

WebKit
Bruce Ong
Nico Popp

WebObjects 1.x - 3.1
Bruce Ong*
Nico Popp*
Charles d'Harcourt**
François Jouaux (joined late in WO 1.x)

Architectural Consultants
Marie Hulot
Bertrand Serlet (acting manager)
Craig Federighi

WebObjects 3.5-4.0
François Jouaux
Charles Lloyd
Eric Bailey
Andrew Belk
Craig Federighi (manager)***

* Left to found RealNames before WO 3.1 shipped
** Left after WO 2.0
*** Left after WO 4.0 for Ariba

a half dozen pages, you definitely will not need WebObjects, but for anyone with complex web needs like those of web publications and e-commerce sites, you won't find software better than the WebObjects/EOF combo at any price.

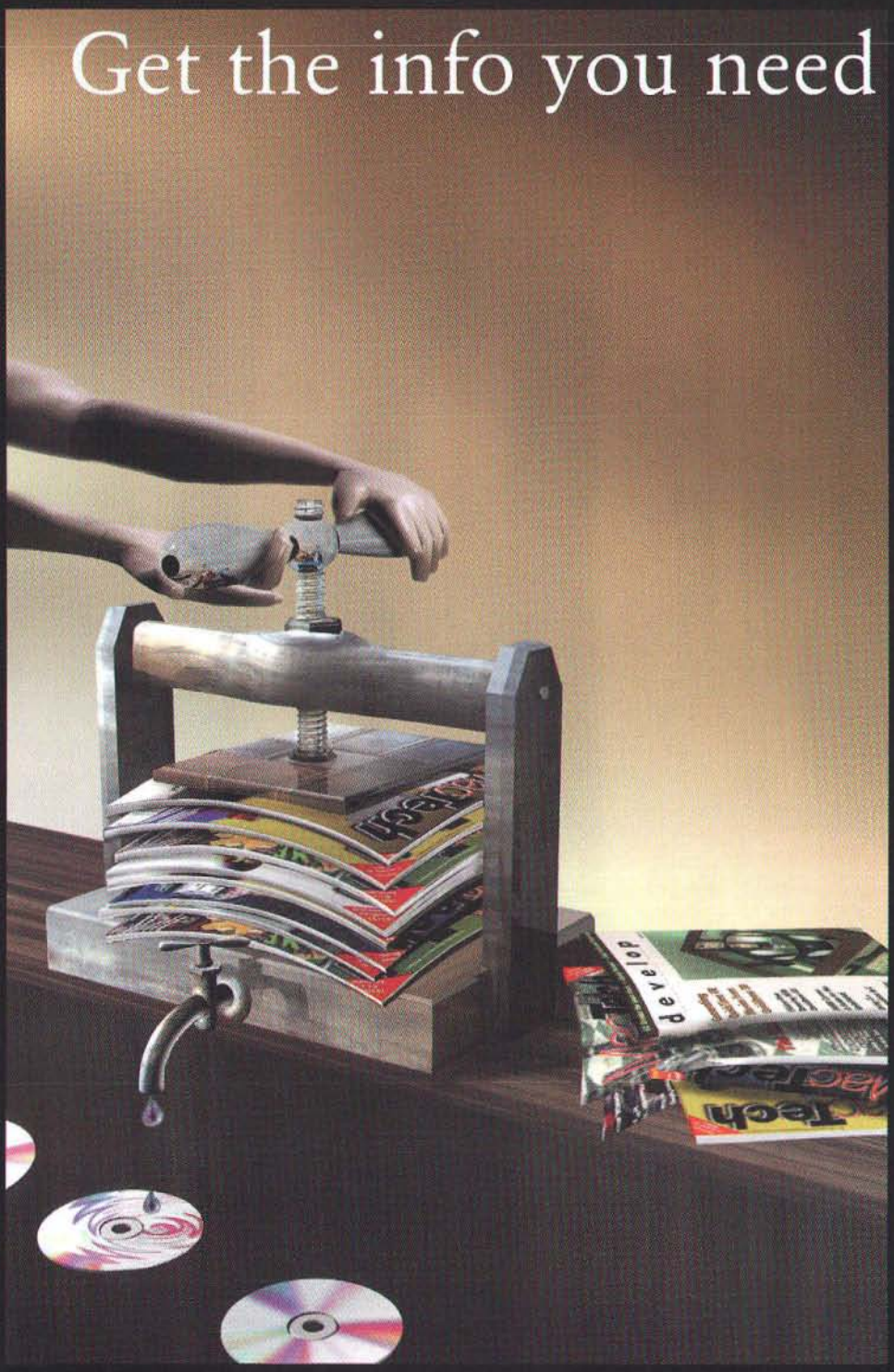
The Web Now Available with the Power of Objects!

WebObjects started its life as a skunkworks project by Bruce Ong and Nico Popp, two quality assurance guys from the EOF and Openstep teams. Bring programmability and database access through an HTML interface was just one of those bright ideas that seems inevitable in retrospect, much like the Web itself. Under the codename "WebKit," management bought into the idea and the team created some basic demos including an application that rendered weather reports for windsurfers.

In its first two versions, WebObjects was built to work with the EOF 1.x stack and had an earlier, more limited form of state management but the outlines of the present architecture began to show. It wasn't until WebObjects 3.0 that the modern EOF 2.x-backed structure which still exists today (in a more refined form) came to be.

One of the goals of WebObjects was to provide as much of the power and flexibility of a desktop application over the Internet. Unlike desktop apps which respond to peripheral events, WebObjects apps respond to HTTP requests, which creates some limitations due to the stateless nature of the protocol (1).There are some basic generalizations one can make about the behaviour of a web transaction: a browser will initiate a request from a particular server, and that server will respond with either the required file or a

MacTech CD-ROM



Get the info you need in one shot!

The MacTech CD-ROM with THINK Reference is the essential reference resource for Macintosh programmers. This CD includes the THINK Reference personal database system and a wealth of Macintosh programming databases, featuring almost 170 issues of the journal of Macintosh programming – MacTech Magazine.This release also features the THINK Reference Compiler, which allows you to compile HTML files into your own compact, searchable THINK Reference databases, and the THINK Reference Collector for building new Mac OS API Databases.

MacTech CD-ROM
with THINK Reference™

PO Box 5200 • Westlake Village, CA • 91359-5200
800/MACDEV-1 (800/622-3381) • Outside US/Canada: 805/494-9797
Fax: 805/494-9798 • E-mail: orders@devdepot.com

message stating that the requested file is unavailable. By hiding behind an HTTP server and communicating through adaptors, a WebObjects-backed application doesn't look any different to a browser than a regular web server would, which is exactly what the designers intended. Other than a funny looking URL, nothing about the documents served from a WebObjects-based site should look different to a browser than a regular static document.

The play-by-play of how a WO application serves a document is quite simple:

1. The user's browser requests a page from an HTTP server
2. The server passes the request on to the WebObjects HTTP adaptor (which can use either CGI or, for performance reasons, a native plug-in architecture like NSAPI for Netscape servers)
3. The WO application calls takeValuesFromRequest() (Java) or takeValuesFromRequest:inContext (Objective-C)
4. The WO application sends a message to the class responsible for managing the page, if any exists
5. After all the appropriate classes have responded, the WO application collects the data to populate the required template, generates a new HTML page and then sends it to the WO Adaptor
6. The WO Adaptor passes the response to the HTTP server
7. The HTTP server returns the page back to the web browser

In most circumstances, you as a developer will only have to write code for steps 3 and 4, the rest of the behaviour is available as the default to the environment. WebObjects is not unique in providing some form of request-response cycle accessible to programmers even among other web application servers. What is different though is that WO requires virtually no programming in order to take advantage of the request-response cycle. In fact, in most circumstances it is unlikely that you'll need to directly intervene at all. This isn't the case for most other application servers which require far more hand-coding, often forcing the developer to write complex spaghetti code (3) for the most common situations.

Making The Web Server Adaptable

When it comes to serving documents WebObjects takes a very agnostic view of the type of web server it prefers. The default adaptor it uses to communicate between itself and the HTTP server is the virtually omnipresent CGI. While common, however, CGI isn't well-suited to high-demand situations because it has to be reloaded for each and every request. So Apple provides different ways of hooking into HTTP servers that allow it. For servers with native plug-in APIs, specialized adaptors are provided to eliminate this potential bottleneck. An interesting result of this design is that the web application doesn't need to be

> Strictly speaking, there are several more types of files associated with a web component. Compiled code specific to a component is stored outside of the page templates in .java (for Java, natch) or .m (for Objective-C) files. If you built the web component using Project Builder, a .api file is also created where you can place any API that should be public to other components. MT

hosted on the same machine (or even run on the same operating system or hardware platform) as the web server it only needs to know where the adaptor is, thereby allowing developers to create sophisticated load-balancing arrangements suited to an application's particular needs.

Since a WebObjects-based site can potentially have multiple applications running at the same time, WebObjects attempts to simplify the process of ferrying information by placing directions within the URL it generates (this is the default, however much of this information can also be stored in cookies, as hidden fields in the HTML or some other customized session management approach). By parsing the URL, the adaptor can tell which instance of which application it should communicate with, the page requested, the requester's session ID and the elementID specified.

For example, parsing this URL tells us some interesting information:

<http://store.apple.com/1-800-MY-APPLE/WebObjects/canadastore.woa/155/wo/FEc0D1WgREnfaBSGZ2/3.5.0.3.27.1>

The site I visited is the Apple Store, but in this case it was running an application called canadastore, which provides Canada-specific prices and information. For performance reasons, the developers set up the site to have multiple instances (the one I was on was number 155). FEc0D1WgREnfaBSGZ2 is the alphanumeric sessionID assigned to that particular transaction. And, 3.5.0.3.27.1 is the element ID assigned to the G4 I was looking at. These session and element IDs are ephemeral though for security reasons and will expire after a developer-determined period of time. When I looked at the site again after closing the window, the site returned the following URL:

http://store.apple.com/1-800-MY-APPLE/WebObjects/canadastore.woa/165/wo/2bHIE0PFaoyshNOI13/2.5.0.3.27.1

As you can see it would be difficult to guess or even read the session information for a particular user making it incredibly difficult to hijack someone else's transaction. This information however is sufficient to overcome the empty, stateless divide between the WebObjects application and the user's browser.

Armed with this session information, the adaptor — whether CGI or one of the others — transmits HTTP requests it receives from the web server and passes them to the WebObjects application instance in a form it can understand. Alternately, on the return trip, the adaptor receives the HTML documents from the WebObjects application instance and passes it on to the web server for eventual delivery to the user's browser. This is the first part of the WebObjects request-response loop.

A Place for Everything, and Everything in its Place

There are several ways of expressing the development philosophy behind the tools Apple inherited from NeXT, but it all boils down to keeping different kinds of code separate. The interface code shouldn't mix with the business logic, which is kept distinct from the data. Part of the reason WebObjects apps are developed so quickly is because it is harder for you as a developer to write spaghetti code which other development environments actually seem to encourage.

Not to bash Bill Gates when he's down, but Microsoft is one of the worst offenders in this tendency to arbitrarily mishmash web application elements. Microsoft Active Server Pages places the HTML template code, the preferred HTTP response, the programming logic, and SQL commands all in a single ASP source file. For developers accustomed to the Model-View-Controller (MVC) approach of WebObjects/EOF and Cocoa, this is an excruciating mess. And if you imagine how much worse it becomes when you start involving database managers, graphic designers and HTML coders in your project, you may come to appreciate the anal-retentiveness of the MVC philosophy.

WebObjects creates these templates or web components using up to 4 types of files: a .html, a .wod, a .woo, a .wos. If you use compiled languages like Java or Objective-C exclusively instead of Apple's interpreted language,Webscript, you'll only have the first three types.

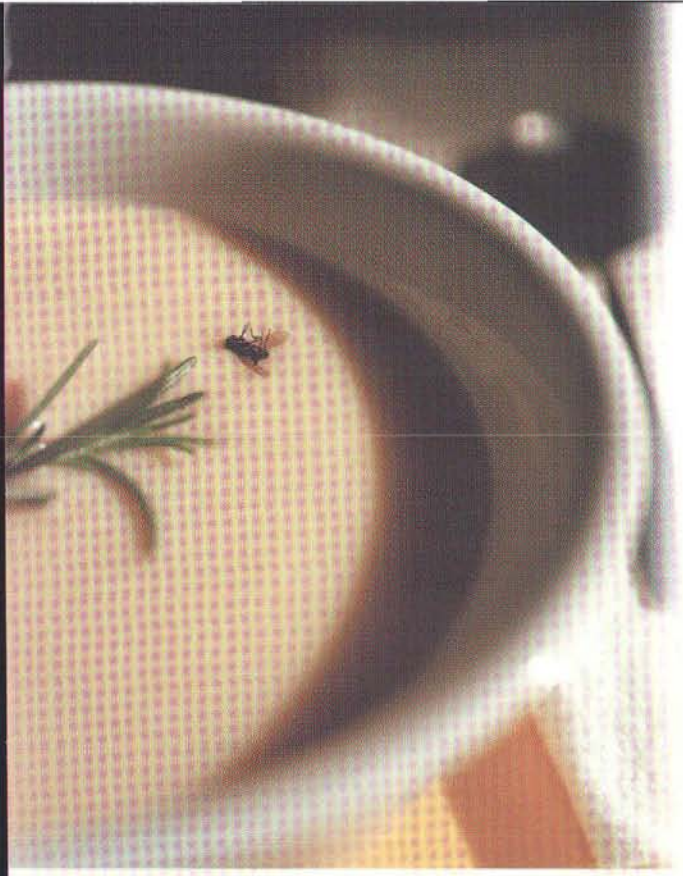
A .html file is the HTML template which collects different components (which can be made up of other HTML fragments) together and provide the shell into which WebObjects structures the data it gets most of the time from EOF. You would be able to recognize a WebObjects specific Form tag by looking for <WEBOBJECT NAME="Form1">...</WEBOBJECT>.

A .wod file is the WebObjects Declaration file, which assigns values to dynamic elements used in the HTML template. They can be either constants or bindings to variables, methods or entities. In the previous example WOForm would be assigned to Form1 to let the application know what type of WO tag it was dealing with. By separating the declarations from the interface, WebObjects makes the job of both the developer and the server easier. The developer is free to modify the interface without lots of declarations littering the code, while the server doesn't have to parse potentially huge files just to find declarations.

You will rarely need to manually edit a .woo file. It is used to setup DisplayGroups and stores information about which version of WebObjects was used to develop the app.

Clean and Simple

WebObjects gives the developers a number of vector points to modifying object behaviour without overwhelming them with choices. If you did want to change the standard request-response loop, there are three places where a developer can intercept and augment it: the Application class, the Session class and the individual WOComponent class for the given page. The appropriate place to intercept the request-response loop will be decided by whether you wish the behaviour to be global to your application, limited to a particular session, or a special case for a specific page. In any case, the Application class will be the first called when the loop is modified following by Session and Component respectively.

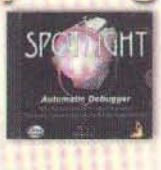


It just takes one bug...

The sooner you find that last bug
the sooner you can ship,
and the sooner you can sleep.

Find these and other essential developer tools at Developer DEPOT®

Spotlight 1.0



only $189!

"Automatic Debugging" on the Macintosh. Easy to use, Spotlight can automatically find run time bugs in your code without your having to change one line of your code. Nail down random bugs immediately. Ever dereference the wrong pointer? Ever pass the wrong value to a toolbox command or over write an array? Your compiler often lets the code compile fine and you may not find that bug until you've burned it into CD-ROM or released it to the net. Find the bugs now, kill them easily, get to sleep sooner.

BugLink Solo

only $79!

The only thing more frustrating that bugs, is bug reports. Users will send in everything from war and peace to "it crashed." BugLink Solo let's you define what information you want in a bug report, then include a customized BugReporter application with your application. Users simply click open the reporter, fill in the boxes, and click send. The report is transparently sent to the email address you defined. BugLink can login into that eMai account, download each report, and present you with an orga-nized, complete, description of each bug. Great for shareware developers, system administrators, and web developers!

DCon 1.0

only $39.95!

Every doctor knows, the secret is listening to the patient. The most frustrating part about debugging on Macintosh is you can't "printf" to show the state of a variable or position in your code. DCon let's your code to talk to you. This system extension adds a console window and file logging services to Macintosh and can be used to record and display status and debugging information during development. It can be called from virtually any code, any where at any time – even from interrupt handlers, I/O completion rou-tines, VBL tasks, deferred tasks, and more! DCon does not allocate memory in any Mac developers debugging arsenal.

www.devdepot.com

PO Box 5200 Westlake Village, CA 91359-5200 • Voice: 800/MACDEV-1 (800/622-3381)
Outside US/Canada: 805/494-9797 • Fax: 805/494-9798 • Email: orders@devdepot.com

There are three methods which can directly interact with the request-response cycle:

Java	Objective-C
`takeValuesFromRequest()`	`-takeValuesFromRequest:inContext:`
`invokeActionForRequest()`	`-invokeActionForRequest:inContext:`
`appendToResponse()`	`-appendToResponse:inContext:`

The method takeValuesFromRequest() is used to process the incoming WORequest object that the WOAdaptor creates immediately after processing the HTTP request the adaptor received from the HTTP server. This method is normally used to get form values and any other inputs that may have been passed through (like a radio button or checkbox selection). By overriding this method, you have considerable control over the request-response loop. [Examples???]

The method invokeActionForRequest() is used to actually call a component action that will return a page. If your application has mandatory fields in a form that a user might neglect to fill out, overriding this method will allow the application to redirect the user to the correct page after verifying that correct information doesn't exist yet. Rather than littering your HTML with conditional responses, the developer can create alternate documents which the application can serve depending on the circumstances.

The method appendToResponse() allows you to change the state of your component after it has been generated but before being relayed to the adaptor. You can perform last minute manipulations of your data before it is sent to the user. A potential use would be if you store some special state tht you don't want to leave in the session, so you could generate a cookie and append it to the response in the overridden method. As well, custom hand-coded HTML could be appended programmatically if you wished.

There are two methods that are important to note:

Java	Objective-C
awake	-awake
sleep	-sleep

The method awake is called every time a component is at the beginning of the request-response cycle. It's normally used by the developer to perform request-handling initialization. This method is different from either the constructor in Java or the -init method in Objective-C. The latter two are called once when the class is first instantiated the very first time the page is returned. The awake method is called whenever a page component begins a new request-response cycle and may also be sent several times during the same request-response cycle to the same component.

The sleep method is usually overridden in order to perform post-request-response processing. For example, you could begin manual garbage collection in Objective-C in the sleep method. As you can probably guess, this method is rarely used except in special cases where you must perform some sort of clean-up duties.

As stated before, most applications require little or no fiddling with the request-response loop. Once data has been pulled in from EOF, WebObjects constructs genuine HTML web pages that it passes on to adaptor. The adaptor relays it to the HTTP server which then transmits it to the browser. And the cycle can begin again.

Footnotes

(1) WWW is perhaps once of a small handful of acronyms which have more syllables than the phrase which it is intended to shorten. This has lead to numerous alternative names like "The Web" or "dub-dub-dub" or, in more confused individuals, "The Internet."

(2) A stateless protocol doesn't know anything about previous transactions. Each request and response cycle is treated as being a self-contained session. For the Web, this is a good thing in some respects, it does provide a degree of privacy to users by requiring extreme efforts on the part of site owners to track sessions and it also reduces the amount of information that must be stored and transmitted during an exchange between the browser and the server. For e-commerce, however, it sucks.

(3) Spaghetti code is a derogatory term for programming done without a coherent structure. While it often occurs when a developer is rushed, much of the worst spaghetti code occurs because the development environment forces the developer to write that way. MT

Want to subscribe to MacTech Magazine?

Need to renew your subscription?

Send an e-mail to orders@mactech.com

Moving?

Remember to update your address by sending an e-mail to custservice@mactech.com

ADVERTISER/ PRODUCT LIST

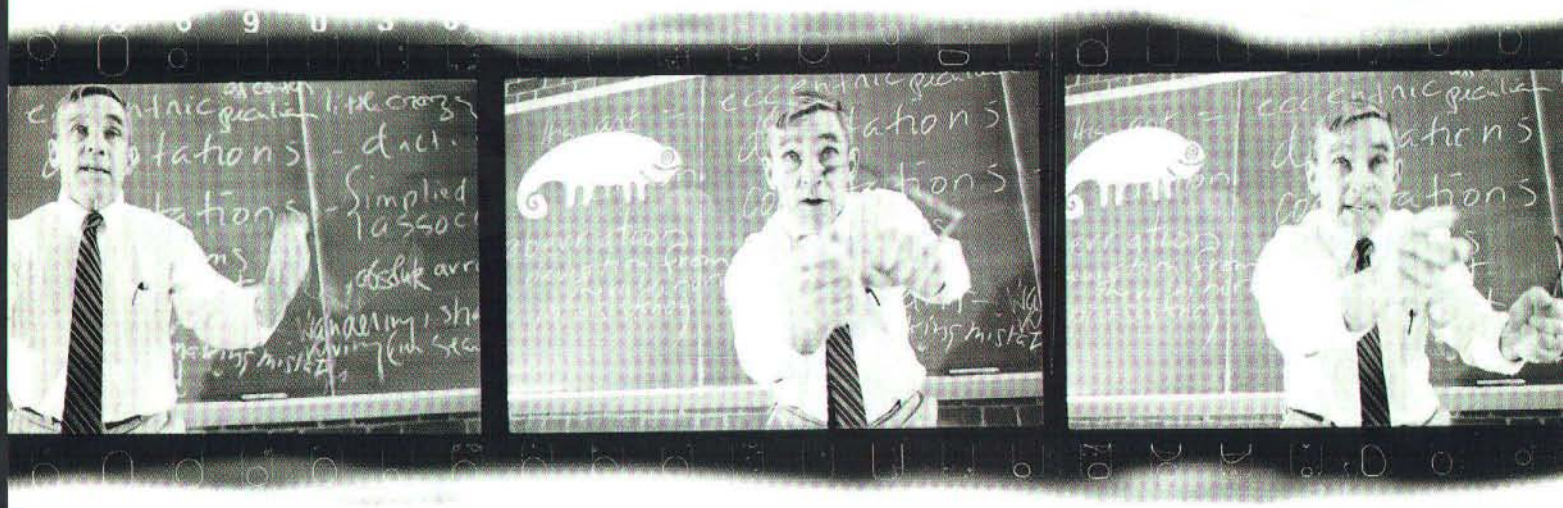
List of Advertisers

List of Products

Mac OS X Porting and Development Showcase

Mac OS X Porting and Development Showcase

HOW DO YOu SAY SuPERB IN LINuX?



SuSE.

{soo´ sah} is {soo´ pûrb}

When you think of Superb, you think of unusually high quality. Like a superb wine, for example. Majestic. Rich. Luxurious. Superior. ■ SuSE Linux is all that. And more. More experience. More adaptability. More applications—over 1500. ■ More power to you. And more freedom, too. ■ No wonder more than 50,000 enterprises worldwide bank on its superb open source solutions. Making SuSE the international Linux leader—setting a higher standard for excellence, simplicity and support. ■ Even the price is superb. ■ So, how do you say superb in Linux? SuSE. It's a lesson well learned.

www.SuSE.com

The freedom to change.
The power to change the Linux world.

SuSE

Versions for Intel, Alpha, and PowerPC

Time matters.

The fate of the company is in your hands.

Every second counts when you're competing at Internet speed. And the faster your Web application is developed, deployed and maintained, the faster your competition will drop out of sight.

The solution? Use Pervasive Software's Tango 2000 to develop your ideas with double the speed of any other development software.

- Visually develop applications on Mac, and deploy on Mac, Windows, Linux and Solaris
- Advanced XML support
- Connect directly to Filemaker or any ODBC database
- Extend applications with JavaBeans
- High performance Tango server scales with cluster support

Give yourself the same competitive edge that Tango has given leading companies like Apple Computer®, theglobe.com™ and Harbor Freight Tools™.

Download your Free Tango Demo today!

Grab the time-saving advantages of Tango 2000 absolutely FREE. Visit our Web site and download our FREE fully functional Tango 2000 demo. Or call us now to get your demo on CD.

www.pervasive.com/speed or **1-800-287-4383**

Hurry! This offer ends soon.

PERVASIVE® SOFTWARE

The Freedom to Create Applications™ for Everyone, Everywhere